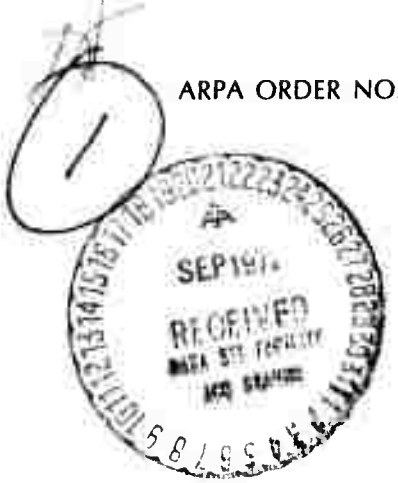
ARPA ORDER NO.: 189-1



R-857-ARPA

April 1972

User's Manual for GLYPLIT: A Program to Translate ILLIAC IV GLYPNIR to IBM 360 PL/I

R. E. Hoffman

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand
SANTA MONICA, CA. 90406

ARPA ORDER NO.: 189-1

R-857-ARPA

April 1972

User's Manual for GLYPLIT: A Program to Translate ILLIAC IV GLYPNIR to IBM 360 PL/I

R. E. Hoffman

A Report prepared for

ADVANCED RESEARCH PROJECTS AGENCY

Rand

SANTA MONICA, CA. 90406

Bibliographies of Selected Rand Publications

Rand maintains a number of special subject bibliographies containing abstracts of Rand publications in fields of wide current interest. The following bibliographies are available upon request:

Aerodynamics • Arms Control • Civil Defense
Communication Satellites • Communication Systems
Communist China • Computer Simulation • Computing Technology
Decisionmaking • Game Theory • Maintenance
Middle East • Policy Sciences • Program Budgeting
SIMSCRIPT and Its Applications • Southeast Asia
Space Technology and Planning • Statistics • Systems Analysis
USSR/East Europe • Weapon Systems Acquisition
Weather Forecasting and Control

To obtain copies of these bibliographies, and to receive information on how to obtain copies of individual publications, write to: Publications Department, Rand, 1700 Main Street, Santa Monica, California 90406.

Published by The Rand Corporation

PREFACE

Before a new computer is installed, it is common practice to debug its programs by simulating the operation of the new machine on an existing computer. GLYPNIR is a programming language for the ILLIAC IV, a computer being developed by the University of Illinois and Burroughs Corporation. To facilitate the debugging of scientific GLYPNIR programs, a GLYPLIT execution process (GLYPNIR to PL/I translator) was devised. The present report is a user's manual for GLYPLIT.

This work was sponsored by the Advanced Research Projects Agency as part of the ARPA/Rand Dynamics of Climate Program, of which a significant element is the use of the ARPA-sponsored ILLIAC IV computer.

This report should be of interest to those facing similar translation problems or considering the differences between code for an ILLIAC IV type machine and a serial machine (especially Appendix A), as well as to users of the GLYPLIT system. The reader is assumed to have a working knowledge of GLYPNIR and IBM's 360 Job Control Language; familiarity with PL/I is desirable but not essential.

SUMMARY

This report is a user's manual for GLYPLIT: a program to translate the ILLIAC IV higher-level language GLYPNIR [1] to PL/I for the IBM 360 [2]. GLYPLIT was developed to provide a mechanism for debugging GLYPNIR programs before the ILLIAC IV computer becomes available, and may still offer a convenient approach to debugging such programs after the ILLIAC IV is operational.

Before a new computer is available, a traditional approach to writing and debugging programs for it is to simulate the new machine on an existing computer. In doing this, at least two programs are usually written for the existing computer: an assembler, which translates assembly language programs for the new computer into the new machine language; and a simulator, which simulates the new machine's operation in this machine language code. The University of Illinois took this approach for the ILLIAC IV with the production of the ASK assembler and SSK simulator for the Burroughs 5500 (and later the 6500). In addition, they produced a higher-level language called GLYPNIR, whose compiler produces ILLIAC IV assembly language for input to the ASK assembler.

Thus potential ILLIAC IV users may write GLYPNIR programs and have them compiled, assembled, and simulated. Unfortunately, the process is quite slow. A typical 1000-card GLYPNIR program takes roughly 2 minutes to compile, 7 minutes to assemble, and, for 3 seconds of ILLIAC IV time, 10 *days* to simulate. If the purpose of a run is hardware simulation, the user must work within this Illinois system; however, if the purpose of a run is to debug a GLYPNIR program, then GLYPLIT offers a faster, more economical approach. With GLYPLIT, the 1000-card, 3-second simulation takes roughly 7 minutes to translate to PL/I, 6 minutes to compile, and 30 *minutes* to execute on a 360/65, a machine comparable to the Burroughs 6500.

GLYPLIT is itself a PL/I and assembly language program that runs on any IBM 360 under the full operating system. PL/I statements may be freely intermixed with the GLYPNIR to be translated, making the full range of PL/I input/output and debugging features available.

The translator does not accept full GLYPNIR as input, but takes a subset suitable for much of the "scientific" programming (FORTRAN-like) expected on the ILLIAC IV. As alluded to above, hardware-oriented features of GLYPNIR are not translatable, such as the ability to insert ASK code, the alphameric WAND and WIMP machine operations, and the explicit specification of registers possible for efficiency. The restriction to scientific GLYPNIR programs also means that all of GLYPNIR's list and pointer processing capabilities have been left out. However, these considerations will not usually interfere with the primary goal of GLYPLIT: to facilitate the debugging of scientific GLYPNIR.

This report details the restrictions of the GLYPNIR subset accepted, describes user control of the translator and insertion of user-supplied PL/I, describes the translator output, and discusses the IBM 360 Job Control Language necessary to run the system. Several appendixes include the structure of the generated PL/I, selected PL/I compiler and execution messages, translator messages, and a summary of the restrictions of the GLYPNIR subset (Appendix F).

ACKNOWLEDGMENTS

Although full responsibility for any errors in the translator program must rest with the author, I wish to thank Duncan Lawrie at the University of Illinois (GLYPNIR's inventor) for numerous conversations and encouragement. Thanks also go to my Rand colleagues Matt Clark and, especially, Sam Oman--GLYPLIT's first, patient user.

CONTENTS

FIGURES

ERRATA

R-857-ARPA USER'S MANUAL FOR GLYPLIT: A PROGRAM TO TRANSLATE ILLIAC IV GLYPNIR TO IBM 360 PL/I
R. E. Hoffman, April 1972

Page 5. Under 2.2.5 Statements, the last sentence should read:

This count does not include leading and trailing blanks on any card, or user-supplied PL/I, or control cards.

Page 16. Mid-page equations should read:

(a(i) DO $i=i_1$ TO i_2) for one subscript,

((a(i,j) DO $i=i_1$ TO i_2) DO $j=j_1$ TO j_2) for two subscripts,

etc. (Of course, i and j may be reversed for the opposite I/O order.)

EX: PUT FILE(SYSPRINT) DATA ((A(I) DO I=1 TO 64)).

Note that two sets of parentheses are necessary--one for the array specification and one for the data list.

Page 33. Line 15.1 should read:

```
15.1   //LKED.SYSLIN  DD DSN=*.PL1L.SYSLIN,DISP=OLD
```

Page 62. A twelfth entry is necessary, which reads:

12. According to GPN/11, GLYPNIR initializes all local variables to zero on the first entry to a block. The translator does not.

1. INTRODUCTION

A traditional approach to writing and debugging programs for a new computer, before that machine is available, is to simulate the new machine on an existing computer. To do this, at least two programs are usually written for the existing computer: an assembler, which translates assembly language programs for the new computer into the new machine; and a simulator, which simulates the operation of the new machine in this machine language code. This approach was taken by the University of Illinois for the ILLIAC IV with the production of the ASK assembler and SSK simulator for the Burroughs 5500 (and later the 6500). In addition, the further step of producing a higher-level language, GLYPNIR [1], was taken. The GLYPNIR compiler produces ILLIAC IV assembly language for input to the ASK assembler.

Thus, potential ILLIAC IV users may write GLYPNIR programs and have them compiled, assembled, and simulated. Unfortunately, the process is quite slow. A typical 1000-card GLYPNIR program takes roughly 2 minutes to compile, 7 minutes to assemble, and, for 3 seconds of ILLIAC IV time, 10 *days* to simulate.†

However, if the purpose of a run is to debug GLYPNIR programs rather than to produce a hardware simulation, then GLYPLIT offers a faster, more economical approach.

GLYPLIT--a GLYPNIR to PL/I Translator--runs on any large‡ IBM System 360 under the full operating system (OS). It accepts a subset of GLYPNIR suitable for much of the "scientific" programming (FORTRAN-like) expected on the ILLIAC IV, and translates it into PL/I [2]. The PL/I may then be compiled and run on any suitable 360, and the results checked. PL/I statements may be freely intermixed with the GLYPNIR to be translated, making the full range of PL/I input/output and debugging features available. For debugging scientific GLYPNIR programs, this GLYPLIT→PL/I→execution process is more economical then the GLYPNIR→

---

†Assuming a simulator ratio of 300,000 seconds of 6500 time to 1 second of ILLIAC IV time.

‡At least 300K of core is required for most GLYPNIR programs. More may be required for syntactically complex programs.

assembly→simulator process. The 1000-card, 3-second simulation takes roughly 7 minutes to translate to PL/I, 6 minutes to compile, and 30 *minutes* to execute on a 360/65.[†]

It must be remembered that the purposes, and thus the results of the simulator and the translator are different. For example, because of internal number representations, answers via GLYPLIT may differ as early as the fourth significant digit from answers via the simulator. Also, hardware-oriented features of GLYPNIR are not translatable, such as the ability to insert ASK code, the alphameric WAND and WIMP machine operations, and the explicit specification of registers possible for efficiency. The restriction to scientific GLYPNIR programs means that all of GLYPNIR's list and pointer processing capabilities have been left out. However, these considerations will not usually interfere with the primary goal of GLYPLIT: to facilitate the debugging of scientific GLYPNIR. (Appendix F summarizes the important restrictions).

1.1 SUGGESTIONS FOR USING THIS MANUAL

Many instructions and explanations in this manual make sense only in the context of a GLYPNIR program using the indicated construct. Thus the reader should not be troubled if all is not clear on the first reading. Furthermore, a number of sections are intended as supplements to the GLYPNIR Programming Manual, and so a complete description of GLYPNIR is not provided. This is especially true of Sec. 2, Restrictions and Differences of the GLYPLIT Subset.[‡]

The best approach for most new GLYPNIR programmers will be to read the GLYPNIR Programming Manual thoroughly, read this manual (especially noting restrictions and differences as per Sec. 2), and then try something. Given the context of a user's program, both manuals will (hopefully) make sense.

[†] A machine comparable to the Burroughs 6500.

[‡] Section 2 is numbered to correspond to the GLYPNIR manual, and sections for which no comment is necessary are not included. Thus to see if there are any translator restrictions for a particular GLYPNIR construct, the user may scan the Sec. 2 subheadings in the table of contents. If the item does not appear, then no special precautions are necessary.

Also note that Fig. 3 of Appendix E is a complete example, including both a GLYPNIR program and Job Control Language (JCL). With minor modifications to the JCL for the user's installation, this example may be punched and run as an aid to getting started.

1.2 ORGANIZATION OF THE REPORT

This report details the restrictions of the GLYPNIR subset accepted (Sec. 2), presents the PL/I input/output and debugging features (Sec. 3), describes user control of the translator and insertion of user-supplied PL/I (Sec. 4), discusses the IBM 360 Job Control Language necessary to run the system (Sec. 5), and describes the translator output (Sec. 6). Several appendixes include the structure of the generated PL/I, PL/I compiler and execution messages, translator messages, and a summary of the restrictions of the GLYPNIR subset.

2. RESTRICTIONS AND DIFFERENCES OF THE GLYPLIT SUBSET

2.1 INTRODUCTION

This section describes the restrictions on full GLYPNIR that define the subset of GLYPNIR acceptable to the GLYPLIT translator. The main omissions are pointers (and thus field-content designators), and such hardware-oriented features as alphameric operators, code statements, and explicit use of hardware registers. (See Appendix F for a summary.)

2.1.1 Numbering of Subsections

In this section, subsection numbers correspond to numbers in the GLYPNIR Programming Manual (GPM) [1]. For example, subsection 2.2.6 corresponds to GPM 2.6, and 2.6.3.2 corresponds to GPM 6.3.2. Those parts of the GPM for which no comment is necessary, i.e., for which there are no restrictions on or differences between GLYPNIR and GLYPLIT, are not included.

An important difference, which applies throughout, is due to the different internal representations of numbers between the ILLIAC IV and the 360 (see 2.3 for details).

2.2 CODING PROCEDURES AND PROGRAM STRUCTURE

2.2.1 Character Set and Coding Form

The 360 GLYPLIT implementation uses the same character set as GLYPNIR,† with the following important exceptions:

GLYPNIR	360 GLYPLIT	USE
[ ]	< > or ¢ !	Subscript delimiters. Note that this may cause problems if an attempt is made to use "<" for "less than." "LSS" should be used instead.‡
×	*	Multiplication symbol.

†But punched in standard 360 EBCDIC instead of B6500 BCL.

‡¢ and ! are the 360 interpretations of [and] punched on a Burroughs EBCDIC keypunch.

GLYPNIR	360 GLYPLIT	USE
←	= or :=	Assignment statement replacement operator.
≤	LEQ	Relational operator.
≥	GEQ	Relational operator.
≠	NEQ	Relational operator.
<	< or LSS	See [ ] above.

The source margins for GLYPLIT are defined by user parameters LC and RC, but are usually defaulted to columns 1-72 (see 4.1.7). No sequence field checking is done.

2.2.4 Identifiers

Only the first 31 characters of any identifier are retained. (ERROR message 48 is generated for identifiers of length > 31 characters.) All of GLYPLIT internally generated identifiers start with the letters 'QQ'. Thus, the user should avoid identifiers starting with 'QQ', except to achieve special effects in user-written PL/I code (see Sec. 4).

2.2.5 Statements

Statements input to GLYPLIT may be split over several cards. The maximum number of characters permitted between semicolons is ∿ 256. This count does not include leading and trailing blanks on any card.

2.2.10 Program

If a label is prefixed to the first BEGIN of the program, it appears at the top of each page of SYSPRINT output. There may be no comments or PL/I after the final 'END.'.

2.3 ILLIAC IV ARCHITECTURE VERSUS 360 ARCHITECTURE

This section covers internal number representations.

GLYPNIR Type INTEGER

	ILLIAC IV	360
number bits	sign + 48	sign + 31
number sig. digits ≈	14.3	9.2
decimal range	±281 474 976 710 655	±2 147 483 647

GLYPNIR Type REAL

	ILLIAC IV	360
Exponent		
number bits	15(base 2)	7(base 16)
decimal range ≈	- 4933 to 4931	- 79 to 75
Mantissa		
number bits	sign + 48	sign + 24 or 56†
number sig. digits ≈	14.3	7.2 or 16.7†
decimal ranges ≈	$\pm 4.2\times10^{-4933}$ to $\pm 5.9\times10^{4931}$	$\pm 5.4\times10^{-79}$ to $\pm 7.2\times10^{75}$

2.4 BASIC ELEMENTS OF THE LANGUAGE

2.4.1 Declarations

Only types PE INTEGER, PE REAL, CU INTEGER, CU REAL, and BOOLEAN (and their abbreviations) are recognized (see 2.9.1 for special subroutine parameter declarations of CNPOINT and PCPOINT).

2.4.2.3 Equivalence on Simple Variables. Not implemented.

2.4.4 Partial Word Designators

Not implemented.

2.4.5.1 Arithmetic Literals. The only acceptable explicit base specifier is '(16)', base sixteen, usually used for BOOLEAN mode patterns.

†Single or double precision as a user option (see 4.2.2).

2.4.5.2 Alphameric Literals. Not implemented.

2.4.5.3 Boolean Literals. See 4.1.8 for effects on Boolean literals when #PES ≠ 64.

2.5 POINTERS, FIELD CONTENT DESIGNATORS, AND DYNAMIC STORAGE ALLOCATION

Not implemented.

2.6 EXPRESSIONS

GLYPLIT accepts only arithmetic and Boolean expressions.

2.6.1 Arithmetic Expressions

GLYPLIT does not accept <arithmetic FCD> or <alphameric expression> as <arithmetic primary>.

2.6.2 Alphameric Expressions

Not implemented.

2.6.3.2 Boolean Literals. Note that only base sixteen '(16)' is legal as an explicit base specifier.

2.7 ASSIGNMENT STATEMENTS

2.7.1 General

Only the real, integer and Boolean entries in Table 7.1.1 in the GLYPNIR Programming Manual are meaningful.

2.8 CONTROL STATEMENTS

2.8.1 LOOP Statement

The <initial>, <increment>, and <limit> values have permissible ranges of -2147483647 to 2147483647 in GLYPLIT. No error messages will be generated unless the values are outside of the above ranges. GLYPNIR's legal ranges are 0 to 1677216 for <initial> and <limit> and 0 to 16384 for <increment>.

2.8.2 FOR Statement (iterative)

Not implemented. The FOR ALL statement (GPM 8.3) *is* implemented.

2.8.7 The Debug Statement

Not implemented.

2.9 SUBROUTINES

2.9.1 Description

Subroutines are as described, except that (1) the AS construct is not legal and (2) empty or missing parameters are not allowed. Note that the formal parameters of the subroutine declaration are the only variables which may be of types PCPOINT or CNPOINT. These types are substitutes for PREAL and CREAL VECTORs. That is, to pass on a real vector (PE or CU) to a subroutine, declare the formal parameter to be of type PCPOINT or CNPOINT, and use the unsubscripted vector name as an actual parameter in the CALL statement. Inside the subroutine, refer to the pointer type parameter as if it is a vector in the usual way. For example:

```
BEGIN
PREAL VECTOR V[10];
SUBROUTINE S(PCPOINT A);
BEGIN
    CINT I;
    LOOP I=1,1,10 DO
        A[0] = A[0] + A[I];
END;
    .
    .
    .
S(V);
    .
    .
    .
END.
```

Note that while inside the subroutine the vector will always be assumed to be of type REAL. If an integer vector is passed to the subroutine, it will be converted to type REAL by the normal parameter-passing mechanism.

Function subroutines may be of type PE or CU REAL or INTEGER or BOOLEAN. GLYPLIT prefixes all function names with the letters 'SS'. See 2.9.3, 4.2.2, A.3, B.1, and B.2 for the effects of this.

2.9.3 Separately Compiled Subroutines

This is a translator feature not yet available in GLYPNIR. Its purpose is to allow subroutines from a GLYPNIR program to be translated and compiled separately, and then link edited and run together (see 5.3 for link edit JCL information). There are at least two possible reasons for using this option. First, the current PL/I implementation allows only 255 iterative DO-END or BEGIN-END blocks in one compilation. PL/I issues TERMINAL ERROR IEM0071I message if this limit is executed, as it may well be for GLYPNIR programs of more than a few hundred cards. Thus, the GLYPNIR program must be broken into two or more parts, which may then be translated and compiled separately. A second reason for using this option is reduced cost, achieved by eliminating the need to retranslate and recompile sections of the code that are not changing.

The only convenient way to allow separate compilation is to have a main program, and one or more separate subroutines. Two problems must be solved: the main program must know about the subroutine (accomplished with the %%C SUBR option), and the separate parts must be able to communicate (accomplished via parameters and/or the %%C EXTERNAL option). Figures 2.1 to 2.3 (and following notes) show a subroutine within a main program, and then the same subroutine and main program ready to be translated and compiled separately.

Note the following:

1. In Fig. 2.1, the P2 inside the subroutine is the same as the P2 outside the subroutine because P2 was not redeclared in the subroutine. Thus, communication is established through the global variable P2, and through the parameter P1.

2. In Fig. 2.2, the main routine is made aware of the subroutine via the %%C SUBR card, followed by the subroutine declaration, but without the body of the subroutine.

3. In Fig. 2.3, just the subroutine is input to the translator. It must now end with "END." instead of "END;", because all input to the translator is terminated by "END.".

Together:

```
BEGIN
PREAL P2;
SUBROUTINE S(PREAL P1);
     BEGIN PREAL A;
           .
           .
           .
       P2 = P1 + A;
           .
           .
           .
     END;


           .
           .
           .
S(...);
           .
           .
           .
END.
```

Fig. 2.1--Main Program and Subroutine Together

Separate: Figures 2.2 and 2.3 are each to be presented to the translator-compiler separately.

```
BEGIN
%%C EXTERNAL
PREAL P2;
%%C SUBR
SUBROUTINE S(PREAL P1);
     .
     .
     .
S(...);
     .
     .
     .
END.
```

Fig. 2.2--Main Program

```
SUBROUTINE S(PREAL P1);
BEGIN PREAL A;
%%C EXTERNAL
PREAL P2;
     .
     .
     .
   P2 = P1 + A;
     .
     .
     .
END.
```

Fig. 2.3--Subroutine

4. The global variable P2 is given the EXTERNAL attribute via the %%C EXTERNAL card in both Fig. 2.2 and Fig. 2.3. This makes it common to all routines in which it is made external. Thus, communication is still through the parameter P1, and the global variable P2.

5. All variables used in the subroutine must be in one of three classes:

   a. They may be parameters passed through the calling sequence, e.g., P1;
   b. They may be global variables known to several routines via the external attribute, e.g., P2;
   c. Or they may be completely internal to the subroutine, e.g., A.

6. Even if no parameters are present, the %%C SUBR card is still required in Fig. 2.2.

7. See 5.3 for JCL information. Sections 4.2.2 and 4.2.4 detail the %%C SUBR and EXTERNAL option.

8. PL/I generates error message IEM2867I (see B.2) for separately-compiled subroutine or external variable names of more than seven characters. To avoid this, limit all separately-compiled subroutine or external variable names to seven characters and, because GLYPLIT prefixes all function subroutine names with 'SS', limit all separately-compiled function subroutine names to five characters.

2.10 SYSTEM SUBROUTINES

2.10.1 SHIFT and REVOLVE

The <value> to be shifted or revolved must be a Boolean expression, e.g., the word MODE.

2.10.2 Route

As noted in the GPM, the <mode pattern> may be any Boolean expression, or empty. Even though GLYPLIT accepts either, *only the second case, empty, is correctly implemented.* Any Boolean expression is ignored. GLYPLIT translates all routes as demonstrated by the following example:

```
PREAL  P1,P2,P3,P4;
P1 = RTL(1,,P2+P3) + P4;
```

becomes

```
DCL (P1,P2,P3,P4)(0:63) FLOAT BIN;
DO QQ=0 to 63; IF MODE (QQ) THEN DO;
    P1(QQ) = P2(MOD(QQ+1,63)) + P3(MOD(QQ+1,63)) + P4(QQ);
```

Thus, the expression is evaluated in the source PE only if the mode is true in the destination PE. This is exactly what GLYPNIR does if the <mode pattern> is empty.

The purpose of a non-empty mode pattern is to prevent evaluation of the expression in certain source PEs, even if the destination PE is true. This results in an undefined value in those destination PEs, but this is presumably to be changed in a subsequent statement. An example might be:

```
P1 = RTL(1,P3 NEQ 0, P2/P3)
IF   RTL(1,,P3) EQL 0 THEN P1 = P2;
```

Unfortunately, the first statement, translated by GLYPLIT, could result in a ZERODIVIDE error, because P2/P3 might be evaluated in a PE, where P3 is zero. An acceptable alternative would be:

```
IF  RTL(1,,P3) NEQ 0 THEN P1=RTL(1,,P2/P3)
ELSE P1=P2;
```

(Note: If the current mode pattern is all true, then "MODE" is frequently used as the routing <mode pattern> because GLYPNIR generates the most optimal code. But no problem can result even though GLYPLIT ignores this, because the mode pattern is all true.)

2.10.4 ALPHA

Not implemented.

2.10.5 BOOLEAN

The actual parameter for the BOOLEAN function *must* be a hexadecimal literal, e.g., BOOLEAN(OFFF(16)).

2.10.6-2.10.8 Trigonometric, Square Root, Natural Log, and Exponentiation

These functions are automatically available in GLYPLIT. In other words, the instructions of GPM 10.6 for including ASK assembly language subroutines must be ignored. No declarations of any kind are necessary or allowed for SIN, COS, TAN, COT, ATAN, SQRT, LN, or EXP.

2.11 INPUT/OUTPUT STATEMENTS

Not implemented. See 4.2 for GLYPLIT I/O via PL/I.

2.12 EXPLICIT USE OF HARDWARE REGISTERS AND ASSEMBLY LANGUAGE

Not implemented.

2.C DIAGNOSTIC MESSAGES

See Appendix C.

2.D OPERATION MANUAL

All cards starting with the '$' in column 1 are ignored by GLYPLIT, except for listing. See Secs. 4 and 5 for GLYPLIT operation.

3. PL/I INPUT/OUTPUT AND DEBUGGING FEATURES

3.1 INTRODUCTION

The ability to mix PL/I statements freely with the GLYPNIR to be debugged is a useful feature of the translator; however, it does require some user knowledge of PL/I. The essentials of PL/I's I/O and debugging features are covered here, but for complex I/O, variable setting, or other PL/I procedures the user is referred to the IBM PL/I manuals [2-3].

3.2 MIXING OF PL/I AND GLYPNIR; THE %%B OPTION

All PL/I statements must be placed on %% cards (%% in columns 1 and 2, and possibly a "B" in column 3). These will appear as comments to GLYPNIR, and will be passed directly to the PL/I file by the translator after stripping off the %% (or %%B). PL/I statements are terminated by semicolons (";"). Several statements may be placed on one card or one statement continued on several cards, much as in GLYPNIR.

The %%B Option: the major task of the translator is to "put in the inner DO loops". Thus the GLYPNIR

```
PREAL P1, P2, P3;
P1 = P2 + P3;
```

must be changed to the PL/I

```
DCL(P1,P2,P3) (0:63) FLOAT BIN;
DO QQ = 0 TO 63;
   IF MODE(QQ) THEN DO;
   P1(QQ) = P2(QQ) + P3(QQ);
END;  END;
```

The %%B option is used to control the location of the user-supplied PL/I. If the "B" is in column 3, i.e., "%%B", then the PL/I on the card is guaranteed to be outside (to Break) any current generated DO loop. If the "B" is not present, the PL/I may be inside a DO loop, if one is currently open. There will be a current open DO loop if one has been started by the appearance of a PE or BOOLEAN

left-hand-side assignment statement, and if all subsequent statements have been PE or Boolean assignment statements and have not involved certain types of routing. That is, assignment statements are collected into a common DO until a non-conforming statement causes the DO to be broken (closed).

There are also other reasons for closing a DO, and it is admittedly harder for a user to be sure his PL/I will end up *inside* a DO. However, this is *not* the common case--being *outside* the DO is usually desired. This is guaranteed by a %%B. The user may of course check the location of his PL/I by reading the PL/I on the SYSPRINT file from the PL/I compiler.

Note that if user PL/I is to be included in a generated DO for special effects, then the DO loop index, always called QQ, may be used. See Appendix A for other naming conventions and more information on DO location.

3.3 PL/I INPUT/OUTPUT

The following is a brief introduction, but should allow the inexperienced user to get started (just try something). Further details are in Ref. 2.

All GLYPNIR I/O must be conducted via PL/I I/O statements. These are GET and PUT for stream reads and writes (corresponds somewhat to FORTRAN formatted I/O), and READ and WRITE for record-oriented I/O (corresponds somewhat to FORTRAN binary or unformatted I/O).

When reading/writing arrays, note that PL/I stores arrays in opposite order to FORTRAN, i.e., the *rightmost* subscript varies fastest in PL/I.

```
{ READ  }
{       } FILE (file-name)  INTO(one-variable-name);
{ WRITE }
```

The READ/WRITE syntax is fairly self-evident. The length of the variable in bytes should usually be the same as the record length (LRECL) of the DCB parameter on the DD JCL card defining the file.

$$\left\{\begin{matrix}\text{GET}\\\text{PUT}\end{matrix}\right\}\ \text{FILE(file-name)}\ \left\{\begin{matrix}\text{DATA (optional-list-of-variables)}\\\text{LIST (list-of-variables)}\\\text{EDIT (list-of-variables)(format specification)}\end{matrix}\right\}$$

The 'FILE(file-name)' is optional, although if omitted, a warning message will be generated during compilation. If it is omitted, file SYSIN will be assumed for a GET, and file SYSPRINT assumed for a PUT.

The format-specification applies to EDIT only. The elements of the list of variables must be separated by commas. Each element may be a simple variable name, an array name, a repetitive specification for arrays (see below), or for output, a constant. Character constants are contained by single quote marks (') and may be split across cards with column 72 and column 1 being taken as contiguous (although note that the %% required in columns 1 and 2 will become blanks in the midst of the constant). A repetitive array specification is of the form

(a(i), DO i=i_1 TO i_2) for one subscript,

((a(i,j), DO i=i_1 TO i_2), DO j=j_1 TO j_2) for two subscripts,

etc. (Of course, i and j may be reversed for the opposite I/O order.)

LIST-directed I/O transmits just the elements. For input the elements must be constants separated by commas or blanks; for output they will be separated by blanks and formatted automatically, according to their attributes.

DATA-directed I/O transmits the variable names as well as the values (like NAMELIST I/O in FORTRAN) and is extremely useful for debugging output. For input, the form is variable-name = constant, separated by commas and/or blanks. The last element of the list must be followed by a semicolon. The list-of-variables is optional: on input, the names in the input determine the variables transmitted, and on output all variables known in the block will be transmitted if no list is given.

EDIT-directed output is most like FORTRAN formatted I/O and requires the format-specification. Format items are separated by commas and may be grouped by placing several in parentheses. A single item or group may be repeated by preceding it with a constant or variable

followed by a blank. Some legal format items and their FORTRAN equivalents follow:

PL/I	FORTRAN	Meaning
F(w)	Iw	Integer in field width w.
F(w,d)	Fw.d	Floating in field width w with d digits to right of decimal point.
E(w,d)	Ew.d	Scientific notation in field width w with d digits to right of decimal point.
A(w)	Aw	Characters in field width w. If w not supplied, length of element being transmitted determines length.
X(w)	wX	w spaces.
SKIP(n)	//.../ (n)	Skip n records--see below.
PAGE	1H1	Skip to top of page.

Note that there is no "Hollerith" format. To output a character constant in PL/I, the constant is placed in quotes in the variable list and an A format specification is used (usually without the field width w).

SKIP and PAGE may be added to any GET/PUT statement (except PAGE and GET are not legal together). The following example illustrates much of the above:

```
PUT FILE(SYSPRINT) EDIT('TWO INTEGERS:',I,J,'FOUR REALS:',
A,B,C,D)(A,F(5),F(10),SKIP(2),A,4 F(10,5))PAGE;
```

LIST-, DATA-, and EDIT-directed I/O is called *stream* I/O because little cognizance is taken of card, line, or other record boundaries. Successive PUT statements will continue on the same line until there is no more room, or until a SKIP or PAGE specification is encountered. Successive GET statements will continue reading from the same card until no more columns remain, or until a SKIP skips to the start of the next card.

3.4 PL/I DEBUGGING FEATURES

The principal debugging aid in PL/I (besides easy LIST- and DATA-directed I/O--see 3.3) is the condition prefix. It looks somewhat like

a GLYPNIR label with the form

(a,b,c,...):

and may be placed on any statement including a BEGIN-END or PROCEDURE-END block. On a block, a condition applies to all statements within the block, except that it may be negated for a statement (or block) within the block by placing the opposite condition on that statement (or block). On an IF statement, the condition applies only to the Boolean expression of the IF. Similarly, a condition on an iterative DO (generated as a result of LOOP and FOR statements) applies only to expressions in the DO statement (j,k, and m in DO i=j TO k BY m) and not to statements contained within the DO-END group.

To cause a condition prefix to apply to a group of statements, enclose the group in a BEGIN-END block and place the prefix on the BEGIN. Condition prefixes may also appear on PROCEDURE statements *(before the procedure name)* to apply to all statements in the procedure. Thus, *to have a condition prefix apply to a whole GLYPNIR program*, make it the very first card presented to the translator (with %% in columns 1 and 2).

If a condition prefix occurs on a labeled statement (or procedure), it must *precede* the label (or procedure name), e.g.,

```
%%(NOOFL,CHECK(A,B)):
   LABEL1:  BEGIN...
```

The content of the condition prefix is a list selected for the following possibilities (there are others described in the PL/I manual [2], but these should be the most useful for a GLYPLIT user). All conditions may be preceded by "NO" to negate them. The conditions are shown with their defaults, i.e., with or without the NO as assumed by the compiler. Also shown are the acceptable abbreviation and the standard system action.

Condition	Abbr.	Meaning
FIXEDOVERFLOW	FOFL	Occurs when an integer exceeds the maximum shown in 2.3. System action· terminate.

Condition	Abbr.	Meaning
OVERFLOW	OFL	Occurs when the magnitude of a real number exceeds the maximum shown in 2.3 (approximately 7.2×10^{75}). System action: terminate.
UNDERFLOW	UFL	Occurs when the magnitude of a real number is less than the minimum shown in 2.3 (approximately 5.4×10^{-79}). System action: set result to 0, print message, and continue.
ZERODIVIDE	ZDIV	Occurs when attempt is made to divide by 0. System action: terminate.
NOSUBSCRIPTRANGE	NOSUBRG	Occurs when attempt is made to use a subscript out of the range of an array. System action: terminate. This can be a useful debugging aid, but because of its great expense (may more than double execution time) it is normally not enabled.
CHECK(a,b,c...)		The most important of the PL/I debugging options. a,b,c... represents a list of names which may be variable names (including unsubscripted array names), entry (procedure) names, and statement labels. For entry names and labels, the name will be automatically printed each time it is invoked or passed. For variables (which may not be parameters), the variable name and its new value will be printed each time it is set, e.g., by appearing as the left-hand side of an assignment statement, by input, or by being the controlled variable of an iterative DO (the result of LOOP and FOR statements, etc.). Two unfortunate notes for CHECK: o CHECK may appear only on a BEGIN-END or PROCEDURE block. o If an array name is CHECKed, the *whole array* will be printed everytime *any* element is changed. Example: assume a PREAL A, which will become an array of length 64 words in the PL/I. The statement "A = 1" will require a DO loop in the PL/I to set each of the 64

Condition	Abbr.	Meaning
		values to 1. If A is being checked, the *whole array* will be printed 64 times, once for each time through the DO loop.

4. USER CONTROL OPTIONS DURING TRANSLATION

There are two classes of options: JCL PARM options, and in-line control options.

4.1 JCL PARM OPTIONS

The user supplies these options on the 360 JCL EXEC statement for the GLYPLIT execution step in the form PARM = 'list', and apply throughout a translation. They help specify either the input format or, more frequently, the output format. Defaults, shown below in parentheses, are supplied for all options. See also 5.1 for details of PARM syntax, and Appendix D for a summary.

4.1.1 #PES--Number of Processing Elements (Default = 64)

This is one of the more important options. It specifies how many PEs the generated PL/I code should simulate. Any number from 1 to 999 is legal. For fast execution of the generated code, debugging may often be conducted using only a few PEs. Alternatively, for duplicating results of existing codes, some number of PEs other than 64 may be useful (e.g., 44 or 100). With most "scientific" GLYPNIR programs, few other changes will be necessary if #PES is changed. In particular, processes depending on the number of PEs and Boolean constants may cause problems (*see 4.1.8 for comments on Boolean constants if #PES ≠ 64*).

4.1.2 GLY--Control of GLYPNIR Output (Default = 1)

If GLY = 1, list the GLYPNIR input only on the SYSPRINT file.

If GLY = 2, output the GLYPNIR input only as PL/I comments on the PLIN file.

If GLY = 3, output the GLYPNIR to both files.

See also the next parameter.

4.1.3 PL1--Control of Generated PL/I Output (Default = 2)

If PL1 = 1, list the generated PL/I only on the SYSPRINT file.

If PL1 = 2, output the generated PL/I only to file PLIN (presumably for input to the PL/I compiler).

If PL1 = 3, output the PL/I to both files.

The most useful combinations of GLY and PL1 are the defaults (GLYPNIR on SYSPRINT, PL/I on PLIN) if the PL/I does *not* need to be looked at, or GLY = 3 and PL1 = 2 (GLYPNIR on SYSPRINT, and GLYPNIR and PL/I mixed on PLIN) if the PL/I *does* need to be looked at.

4.1.4 PAGES--Maximum Number of Pages Allowed (Default = 50)

This parameter, like MSGS, is intended to prevent an error from generating reams of useless output.

4.1.5 MSGS--Maximum Number of Messages Allowed (Default = 50)

This parameter option prevents some runaway translator or user error from generating useless error message output.

4.1.6 MSG--Minimum Significant Message Level (Default = 1)

Messages are classed into four levels: 1--WARNING, 2--ERROR, 3--SEVERE, and 4--TERMINAL. This parameter specifies the minimum level for which messages should be printed. Messages in a level below MSG are neither printed nor do they contribute toward the MSGS count (see 4.1.5). The level of each message is printed with it and is also noted in Appendix C.

4.1.7 LC (Left Column), RC (Right Column), and CC (Carriage Control)--Source Card Margin Control (Defaults = 1, 72, 0 respectively)

Normally GLYPNIR input should be in columns 1-72, with columns 73-80 ignored, except for listing; however, special purposes may require different margins. If so, LC may be set to the leftmost column to be considered, and RC to the right. In addition, CC may be set to a column to be scanned for printer carriage control as follows:

Contents of Column CC	Action
1	Skip to top of page
0 (zero)	Double space

Contents of Column CC	Action
- (minus)	Triple space
+ (plus)	No space
Anything else	Single space

Note that the GLYPNIR compiler does not support these options.

LC. RC, and CC must obey the relations

$1 \leq LC, RC, CC \leq 80$

$LC < RC$

$CC < LC$ or $CC > RC$.

All references to columns 1 and 72 throughout this manual should be interpreted as referring to columns LC and RC, respectively.

4.1.8 BCRC--Boolean Constant Repeat Count (Default = 1)

This section discusses the treatment of Boolean hex constants by GLYPLIT.

Define n as being the number of bits represented by a hex constant; thus, n = the number of hex characters in the constant × 4. *Note that n is always a multiple of 4*, although #PES need not be. Thus n may vary from 4 (1 hex character) to 64 (16 characters). The problem is to map the n bits represented by the constant into the #PES PEs. There are three cases.

4.1.8.1 #PES = 64. This is the normal GLYPNIR case. If n = 64, the n bits map exactly left to right into PEs 0 to 63.

If n < 64, then the n bits map into the n rightmost PEs, and the leftmost 64 - n PEs get zeros.

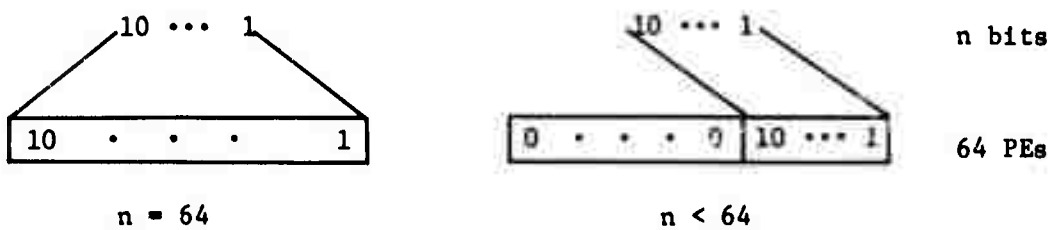


Fig. 4.1--#PES = 64

4.1.8.2 #PES < 64. If n ≤ #PES, the n bits map into the rightmost n PEs and the leftmost #PES - n PEs get zeros.

If n > #PES, then the leftmost #PES/2 bits map into the leftmost #PES/2 PEs, and the rightmost #PES/2 bits will map into the rightmost #PES/2 PEs. (If #PES is odd, then the extra bit is included on the left.) In other words, the n - #PES middle bits are ignored.

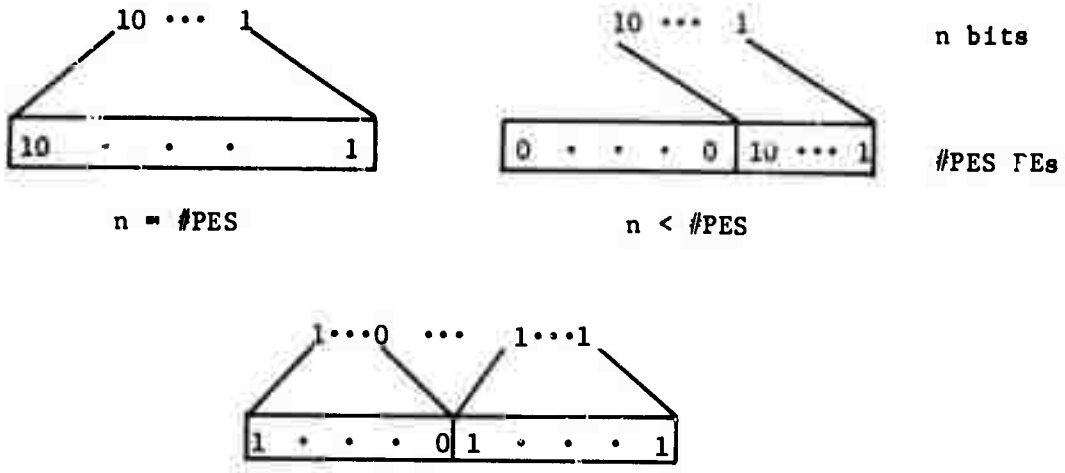


Fig. 4.2--#PES < 64

4.1.8.3 #PES > 64. If n < 64, the n bits map into the rightmost n PEs, and the leftmost #PES - n PEs get zeros.

If n = 64, then the leftmost 32 bits map into PEs 0 to 31, the rightmost 32 bits map into the rightmost PEs, and the middle PEs are filled by repeating bits 33-BCRC through 32 as often as necessary. That is, the rightmost BCRC bits of the leftmost 32 bits are repeated to fill the middle.

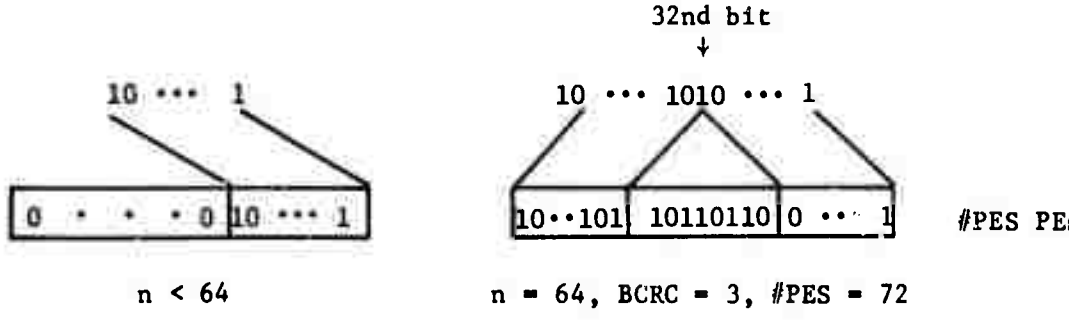


Fig. 4.3--#PES > 64

4.1.8.4 User-Supplied Interpretation of Boolean Constants. If the actions described above for #PES ≠ 64 are inadequate, then there are two alternate approaches a user may take. Whenever a Boolean constant appears, it is replaced by a QQBCx variable, e.g., MODE = BOOLEAN (OF(16)) becomes MODE(QQ) = QQBC1(QQ) within a DO loop. The Boolean constant variables are initialized via the following declaration:

```
DCL QQBCx (0:63) BIT(1) INIT CALL QQBCI(QQBCx,'hc');
```

"x" is an integer to form a unique name, and 'hc' is the actual hex constant character string in quotes. Then, on entry to the program, PL/I automatically calls the procedure QQBCI once for each declaration, which initializes the variable 'QQBCx' to the hex string 'hc', as described in 4.1.8.1-4.1.8.3. (The QQBCI procedure is contained in the GLYPLIT execution library-card 15 of 5.2.) A group of ten of these declarations is generated for every ten unique Boolean constants encountered, with a final group (the only group if there are no more than ten in the program) occurring at the end of the generated code.

The first alternative is to make the user parameter BCRC = 0. In this case, no declaration for Boolean constants will be generated, although each unique Boolean constant will still be replaced by a variable of the form QQBCx. Thus the user is expected to declare and initialize all the QQBCx variables generated via user-written PL/I.

A second alternative is for the user to supply his own routine called QQBCI. It should be of the following form:

```
QQBCI:  PROCEDURE(BC,LITERAL);
DCL BC(0:n) BIT(1), LITERAL CHAR(16) VAR;
        .
        .
        .
END;
```

where n = #PES - 1.

Because the literal will be passed exactly as presented by the user in his GLYPNIR input, the user may simply make his Boolean

constants numbers, e.g., '1', '2', etc., and use LITERAL as an index into a table containing the desired initial values. An example follows for Boolean constants of 3 bits.

```
QQBCI:  PROCEDURE(BC, LITERAL);
DCL BC(0:2) BIT(1), LITERAL CHAR(16) VAR;
DCL TBL(0:2,2) BIT(1) INIT('010','101');

     BC = TBL(*,LITERAL);
END
```

The assignment statement is an array assignment: the '*' means all elements in the range of that dimension. PL/I automatically converts the character contents of LITERAL to integer if the contents are a number.

Thus, if the user supplies this routine, and does not give BCRC = 0 (which would cause no declarations to be generated), then the statement

```
B1 = BOOLEAN(1(16)) OR BOOLEAN(2(16));
```

is translated as (assuming #PES = 3)

```
DO QQ = 0 TO 2;
  B1(QQ) = QQBC1(QQ) | QQBC2(QQ);
END;
```

and causes the declarations

```
DCL QQBC1(0:2) BIT(1) INIT CALL QQBCI(QQBC1,'1');
DCL QQBC2(0:2) BIT(1) INIT CALL QQBCI(QQBC2,'2');
```

to be generated. Then on entry to the main program, QQBCI would be called twice and QQBC1 and QQBC2 would be initialized to '010' and '101'.

4.1.9 LINES--Number of Lines per Page of SYSPRINT Output (Default = 60)

Useful for installations with short paper.

4.2 IN-LINE CONTROL OPTIONS

These options are placed on %%C cards and mixed with the GLYPNIR input. Unlike PARM options, which apply to a whole translation, these options normally apply to only the next statement or to a range of statements.

4.2.1 %%C Cards

All in-line GLYPLIT control options must be placed on cards starting with %%C in columns 1, 2, and 3. (These cards will be comments to GLYPNIR.) Several options may be placed on one card but the option words START and END may not both appear on one card. All option words may appear in any order separated by any number of blanks *or any other characters*. Note that this means that comment cards starting with %%C should be avoided because they will be misinterpreted as option cards. (See 3.2 for discussion of %% cards for PL/I.)

4.2.2 EXTERNAL Declarations

Declarations may be made to have the PL/I external attribute by use of the %%C EXTERNAL card. This card should precede a GLYPNIR declaration for which the external PL/I declaration is to be generated. The option normally applies only to the following declaration. If the option word "START" is also on the card, however, all following declarations will be made external until a %%C option card with the words "EXTERNAL" and "END" appears.

External declarations and subroutine parameters are the only easy methods of inter-subroutine communication for *separately compiled* procedures in PL/I. The external attribute takes the place of COMMON in FORTRAN. (See 2.9.3 for separately compiled procedures.)

Note that PL/I generates an error message (see B.2) for separately-compiled subroutine or external variable names of more than seven characters. To avoid this, limit all separately-compiled subroutine or external variable names to seven characters and, because GLYPLIT prefixes all function subroutine names with 'SS', limit all separately-compiled function subroutine names to five characters.

In conformance with a recently issued ILLIAC IV Document [6], COMMON and ENDCOMMON statements may be used instead of %%C START EXTERNAL and %%C END EXTERNAL, respectively. However, using COMMON and ENDCOMMON will have *exactly the same effect* as the START and END EXTERNAL statements and no more. That is, the <common block name> will be completely ignored (if supplied), and the rule that does not apply is "two variables which occur at the same place in the common block correspond even though their names may be different." As noted, using EXTERNAL and COMMON simply gives the variables being declared the PL/I external attribute. In PL/I, variables with the external attribute in different procedures are matched strictly by name, and it does not matter where the declarations occur in the procedures.

4.2.3 DOUBLE Precision Declarations

The PL/I declarations generated for GLYPNIR declaration statements may be made double precision via the %%C DOUBLE card. (See 2.3 for meaning of single and double precision.) The option normally applies only to the next declaration. If the option word "START" is also on the card, however, all following declarations will be made double precision until a %%C option card with the words "DOUBLE" and "END" appears.

Note that all built-in functions (EXP, SIN, etc.) will automatically return double precision results if their arguments are double precision.

4.2.4 SUBR Declaration

The %%C SUBR causes the next subroutine declaration to generate only the entry information. It is assumed that the body of the subroutine does not follow, but will instead be compiled separately. (See 2.9.3 for full details.) This option normally applies only to the next subroutine declaration. If the option word "START" is also on the card, however, all following subroutine declarations will generate only entry information until a %%C option card with the words "SUBR" and "END" appears.

Also, see note at the end of 4.2.2.

In conformance with a recently issued ILLIAC IV Document [6], subroutine entry declarations may also be made by preceding the declaration with the word EXTERNAL. The <output> and <filename part> do not, however, apply and must not be specified.

5. JOB CONTROL LANGUAGE

This section describes, mainly by example, the IBM 360 JCL necessary to execute the translator, to compile the generated PL/I, to link edit the object code along with the GLYPLIT execution library, and to execute the final result. These steps are called the GLYPLIT translation step, the PL/I compilation step, the link edit step, and the execution GO step, respectively.

The JCL used for the last three steps is the standard, IBM-supplied PL/I Compile-Link Edit-GO cataloged procedure found at most installations. As such, the complete cataloged procedure is not displayed; only the cataloged procedure EXEC card and the override DD cards are shown.

As may be seen by the IBM technical terms already used without definition, this section is not a course in JCL; rather, the user is assumed to have a basic knowledge of IBM JCL and to be capable of modifying what follows for his own needs and installation. For more JCL information, see Refs. 4 and 5. Sample listings may be found in Appendix E.

5.1 GLYPLIT TRANSLATION STEP

```
1 //jobname JOB installation-defined-parameters
2 //GLYPLIT EXEC PGM=GLYPGO,REGION=300K,
3 //   PARM='parameter-list'
4 //STEPLIB DD DSN=library,DISP=SHR
5 //SYSPRINT DD SYSOUT=A,
6 //    DCB=(RECFM=VBA,LRECL=125,BLKSIZE=3129)
7 //PLIN  DD  DSN=plin-file,DISP=plin-disp,
8 //    DCB=(RECFM=FB,LRECL=80,BLKSIZE=7280)
9 //GLYPNIR  DD  *
```

Card 2. The GLYPLIT translator program is named GLYPGO and is assumed to exist in the "library" of card 4. The REGION shown, 300K, should be adequate for most translations, except for those requiring an especially deep parse (e.g., very complex arithmetic expressions). Less than 300K may also be used for economy.

Card 3. The PARM parameter is used to supply the options described in 4.1. Individual parameters are separated by commas (if there are more than one) and the whole list is enclosed in quotes. The PARM parameter is not necessary if all of the defaults shown in 4.1 are acceptable.

Card 4. The "library" should be the installation-defined partitioned data set name containing the translator module, GLYPGO.

Cards 5 and *6.* The SYSPRINT card defines the file for all translator messages output, as well as the GLYPNIR input listing and possibly a list of the generated PL/I as controlled by the PARM parameters GLY and PL1 (see 4.1.2-4.1.3). Other blocking factors may be used.

Cards 7 and *8.* The PLIN card defines the file for the output PL/I generated by the translator. As with SYSPRINT, its contents are controlled by the parameters GLY and PL1. The "plin-file" name may be a temporary (e.g., &&TEMP) or a permanent file name. A permanent file may be especially useful if a text editing system to edit the file is available. Corrections or additions may then be made to the generated PL/I before it is compiled. Alternatively modifications can be made by punching the file as cards. The "plin-disp" will depend on whether the file is old, new, temporary, permanent, etc. If the file is new, UNIT, VOLUME, and SPACE parameters will be required. With respect to SPACE, the translator typically generates 2.2 cards of PL/I for every card of GLYPNIR input (assuming the GLYPNIR input is *not* being copied to PLIN). Other blocking factors may be used.

Card 9. This card defines the location of the GLYPNIR to be input to the translator. The example assumes that the GLYPNIR cards follow in the input stream.

See Appendix C for the condition codes returned by the translator.

5.2 THE PL/I COMPILE, LINK EDIT, AND GO STEP

The IBM-supplied cataloged procedure, PL1LFCLG for PL/I Level F Compile-Link Edit-GO, is used for these three steps. Only the procedure execution card and override DD cards are shown.

```
10  //  EXEC PL1LFCLG,REGION.PL1L=250K,COND.PL1L=(15,LT,GLYPLIT),
11  //    PARM PL1L='SM=(2,72),SIZE=999999,STMT,NEST',
12  //    TIME.GO=go-time,REGION.GO=go-region
13  //PL1L.SYSIN  DD  DSN=plin-file,DISP=plin-final-disp
14  //LKED.SYSLIB  DD  DSN=SYS1.PL1LIB,DISP=SHR
15  //    DD  DSN=GLYPLIT-execution-library,DISP=SHR
16  //LKED.SYSLMOD  DD  DSN=user-pds(mod-name),DISP=OLD
17  //GO.SYSIN  DD  *
```

Card 10. The PL/I generated for GLYPNIR programs of more than a couple hundred cards usually causes the PL/I compiler to be inefficient if the standard compilation region is used. The region shown, 250K, usually results in economical compilation for programs under 1000 cards of GLYPNIR. If the compilation time seems excessive, increase the region; alternatively smaller programs may permit a decreased region. (It takes ∿ 25 CPU sec on a 360/91 to translate 500 typical GLYPNIR cards, and 16 CPU sec to compile the 1100 generated PL/I statements.) The COND.PL1L parameter shown stops compilation if TERMINAL errors were made during translation. Even if SEVERE errors were made, it is frequently worthwhile to be able to see the PL/I generated, although a lower COND parameter may be used. (See Appendix C for condition codes returned.)

Card 11. The SM=(2,72) part of the PL1L.PARM is included mainly to exhibit the source margins of the GLYPLIT generated PL/I; this value is usually the compiler-supplied default. The SIZE subparameter is necessary to take advantage of whatever REGION is supplied on card 10. The STMT subparameter, also usually a compiler default, causes code to be generated so that if the compiled program fails during execution, error messages will include the PL/I statement number. (Appendix B discusses tracing problems back to the GLYPNIR.) The NEST subparameter is very useful for matching BEGIN or DD - END blocks.

Card 12. As always, it is important to provide a limiting "go-time" for the execution step, and usually necessary to provide a "go-region" to override the small standard default.

In addition, it may be desirable to override the PL/I and LKED COND parameters, especially if PL/I messages at the ERROR level are

generated. As noted in Appendix B, some ERROR level messages may not impede execution.

Card 13. The SYSIN card defines the input to the compiler. The "plin-file" name may refer back to or be the same as the PLIN DD-name from card 7, and the "plin-final-disp" will indicate the desired status of the file after the job.

Cards 14 and *15.* The LKED.SYSLIB definition is a concatenation of the standard PL/I library and the installation-defined residence of the GLYPLIT execution library. This library includes all of the mathematical and data manipulation functions built into the GLYPLIT system (e.g., MAX, MIN, ROWSUM) as well as a routine called QQBCI for initializing Boolean constants.

Card 16. The LKED.SYSLMOD card is optionally used to save the final translated and compiled program for repeated execution with different user data. The example assumes an existing partitioned data set within which the link-edited module is to be saved under the name "mod-name".

Card 17. The GO.SYSIN card defines the source of any user data (if present). This card will be required if PL/I statements such as

```
%%B  GET FILE (SYSIN) EDIT(A,B,C)(3 F(10,4));
```

have been inserted with the GLYPNIR. Other DD cards may also be added to the GO step if required by the user.

5.3 SEPARATELY COMPILED SUBROUTINES

This section presents only the additional JCL necessary to translate and compile subroutines separately, followed by a link edit and GO step to run them together. (See 2.9.3 for discussion of this option, with examples.)

The JCL for the GLYPLIT step is the same, except that the "plin-file" name must be changed for each separate translation if the results are to be saved permanently.

Instead of a combined Compile-Link Edit-GO step, each piece is just compiled separately with something like the following JCL:

```
20  //  EXEC  PL1LFC,PARM.PL1L='SM=(2,72),STMT'
21  //PL1L.SYSIN  DD  DSN=plin-file,DISP=plin-final-disp
22  //PL1L.SYSLIN  DD  DSN=object-pds(mod-name_i),DISP=OLD,
23  //      DCB=BLKSIZE=3200
```

Card 20. If the piece being compiled is small, the standard region will probably be adequate.

Card 21. As before, except that "plin-file" must be changed for each separate translation if the output from each translation is being saved.

Cards 22 and *23*. This is the output from the PL/I compiler: an object module named "mod-name$_i$" is saved in an already existing partitioned data set with the name "object-pds". "Mod-name$_i$" is the member name for this module and should be different for each separately compiled routine.

To link edit and run the modules together, a separate Link Edit-GO step may be used, or the final compilation may use the same Compile-Link Edit-GO step shown above. If using PL1LFCLG, the only additional JCL required is a //LKED.SYSLIN data set and a //LKED.OBJLIB card as follows:

```
15.1  //LKED.SYSIN  DD  DSN=*.PL1L.SYSLIN,DISP=OLD
15.2  //  DD *
15.3        INCLUDE OBJLIB(mod-name_1mod-name_2,...)
16.1  //LKED.OBJLIB  DD DSN=object-pds,DISP=OLD
```

The cards are numbered to show where they belong in the PL1LFCLG JCL shown above. All "mod-names" separately compiled and saved on the "object-pds" should be included (the new module just compiled is automatically included and its name, if on the INCLUDE card, will be ignored).

The above is only one of many ways to accomplish the link edit, but it is fairly convenient and short.

6. GLYPLIT OUTPUT

GLYPLIT generates two output files, SYSPRINT and PLIN, under control of the two user parameters GLY and PL1. File SYSPRINT contains all messages issued during translation, as well as a listing of the GLYPNIR input (if GLY=1 or 3), and a listing of the generated PL/I (if PL1=1 or 3). File PLIN is the input to the PL/I compiler. It contains the generated PL/I (if PL1=2 or 3) and the GLYPNIR input as PL/I comments (if GLY=2 or 3). (GLY and PL1 are described in 4.1.2 and 4.1.3.)

6.1 SYSPRINT FILE

Figure 6.1 is a sample of the translator's SYSPRINT output. The date, time, and GLYPLIT version number are shown at the top of each page. Also shown is the name of the routine being translated at the start of the page. For the main routine, the label on the first BEGIN will be shown if it exists. (If not, the name 'MAIN' will be used. The translating name may be blank for the first one or two pages.)

Page 1 of the output shows the PARM options specified by the user as well as a complete list of options in force during the translation.

Subsequent pages include, for each card, the card number, the subroutine nesting level, and the BEGIN-END block level within the subroutine. These last two numbers may run a few cards behind the listing of the card which starts the subroutine or block.

The translator messages format is fully described in Appendix C.

6.2 PLIN FILE

This is a card image file intended to be input to the PL/I compiler. In addition to the PL/I (both generated and supplied by the user on %% cards), it will contain the GLYPNIR input as PL/I comments if GLY=2 or 3. This option is useful during attempts to study both the results of translation and the relationships between generated and user-supplied PL/I. Two caveats: the GLYPNIR statement may appear a few PL/I statements before the actual PL/I statements it caused to be generated; and if several GLYPNIR statements appear on one card, the card will be

```
GLYPLIT     VERSION 1.1        TRANSLATING:                    DATE:  03/02/72     TIME:  11:59:52          PAGE   1


GLYPLIT TRANSLATOR OPTIONS SPECIFIED ARE AS FOLLOWS:

     GLY=3,#PES=72

THE COMPLETE LIST OF OPTIONS FOR THIS TRANSLATION IS:
                                                        #PES= 72
                                                        GLY=3
                                                        PL1=2
                                                        PAGES= 50
                                                        MSGS= 50
                                                        MSG=0
                                                        LM= 1


GLYPLIT     VERSION 1.1        TRANSLATING: LEV1               DATE:  03/02/72     TIME:  11:59:52          PAGE   6

  CARD SUBR BLOCK

  226   1     2                  A = DELT*B ;
  227   1     2                 Q3T<J> = Q3T<J> + A ;
  228   1     2                 Q3T<J-1> = Q3T<J-1> - A ;
  229   1     2              END ;
  230   1     1     END;  % SUBROUTINE LEV1
  231                 SUBROUTINE AVRX(PCPOINT Q);
  232   1             % SMOOTHS ARRAY C
  233   1             BEGIN
  234   1                   PREAL A;
  235   1     1             CREAL ALP,DEFF,DRAT;    CINT NM;
  236   1     1             DEFF = DYP<JM DIV 2 + 1> ;
  237   1     1             LOOP J=1,1,JM1 DO
  238   1     1             BEGIN
  239   1     1                   DRAT = DEFF / DXP<J> ;
  240   1     2                   IF DRAT GTR 1.0 THEN
  241   1     2                   BEGIN
  242   1     2                         ALP = 0.125*(DRAT-1.0);
  243   1     3                         NM = DRAT;
  244   1     3                         LOOP I=1,1,NM DO
  245   1     3                         BEGIN
  246   1     3                               A = Q<J> ;
  247   1     4                               Q<J> = A + ALP*(RTL(1,MODE,A) + RTR(1,MODE,A)
  248   1     4                                           - 2.0*A) ;
  249   1     4                         END;
  250   1     3                   END;
  251   1     2             END;
  252   1     1   %
  253   1     1   %
  254   1     1   END;   % SUBROUTINE AVRX
  255                  SUBROUTINE LEV2(PCPOINT U,PCPOINT V,PCPOINT T2,
  256                                  PCPOINT UT,PCPOINT VT,PCPOINT TT2);
  257   1              BEGIN
  258   1             %          CORIOLIS FORCES TO U AND V
  259   1                   FD<0> = 0.0;
  260   1     1             FD<JM> = 0.0;
  261   1     1             LOOP J=1,1,JM1 DO
```

Fig. C.1 Sample Output

relisted several times (unchanged), once for each statement. (Note that the expressions "IF Boolean expression THEN", "ELSE", "LOOP i_1=i_2,i_3,i_4 DO", "label:", etc., all count as "statements" for multiple card-listing purposes.)

Each PL/I statement contains a sequence identifier in columns 73-80. Columns 73-76 contain the first four characters of the label on the first BEGIN (if it exists) or "MAIN" (if it does not). Columns 77-80 contain a (probably non-unique) sequence number. The same sequence number is given to all PL/I statements generated (directly or indirectly) by a single GLYPNIR statement, and is equal to the card number of the GLYPNIR statement. When an error occurs during execution of the translated program, the system error message will usually include the PL/I statement number. By referring to the SYSPRINT listing generated during the PL/I compile step, the statement number may be associated with the sequence number, which leads to the GLYPNIR card number shown on the listing from the GLYPLIT step.

See Appendix A for a general description of the PLIN file and naming conventions followed.

Appendix A

GENERATED PL/I STRUCTURE

The purpose of this appendix is to aid those users who must study the PL/I generated on the file PLIN. A discussion of overall structure is followed by notes for individual statement types, and then the naming conventions employed.

OVERALL STRUCTURE OF GENERATED PL/I

The first BEGIN causes the generation of the statement "label: PROCEDURE OPTIONS(MAIN);" where the label is the label on the BEGIN (if it exists) or 'MAIN' (if there is no label). Following this are always a large number of DCL statements for MODE, PEN, the various built-in functions, etc. Then comes the bulk of the user's program, and finally additional DCL statements for Boolean constants, the IF Boolean expression stack, formal and actual parameters, and any other variables that the translation may require in addition to the user's variables.

NOTES FOR SELECTED INDIVIDUAL STATEMENT TYPES

Assignment Statements and Generated DO Loops

The translator has two main tasks. The first is to change GLYPNIR's syntax to PL/I's. Thus, LOOP I=1,1,5 DO becomes DO I=1 TO 5 BY 1. The second and major task is to "put in the inner DO loops". Thus

```
PREAL P1,P2,P3;
P1=P2+P3;
```

becomes

```
DCL (P1,P2,P3)(0:63) FLOAT BIN(21);
DO QQ=0 TO 63;
   IF MODE(QQ) THEN DO;
   P1(QQ)=P2(QQ)+P3(QQ);
   END;  END;
```

It is clearly desirable to collect as many assignment statements as possible into the same DO, thereby reducing the overhead associated with the DO and IF MODE(QQ) statements. Assignment statements are collected into the same DO as long as they do not have a CU lefthand side, and do not have any routing conflicts with assignments already in the DO. An example of this last condition would be the following:

```
PREAL P1,P2,P3;
P3 = RTR(1,,P1);
P1 = P2
```

RTR(1,,P1) is translated to P1(MOD(QQ-1,63)). If the two assignment statements are collected in the same DO, then P1(QQ) will be updated by the second assignment before it is used by the first.

If the routing conflict occurs in the same statement, then a temporary lefthand side is needed. Thus

```
PINT P1;
P1 = RTL(1,,P1);
```

becomes

```
DCL P1(0:63) FIXED BIN;
DO QQ=0 TO 63; IF MODE(QQ) THEN DO;
   QQAST(QQ)=P1(MOD(QQ+1),63);
   END;  END;
DO QQ=0 TO 63; IF MODE(QQ) THEN DO;
   P1(QQ)=QQAST(QQ);
   .
   .
   .
```

The models for arithmetic assignment statements and routing have already been given by example above. The model for Boolean assignments, shown by example below, are different from arithmetic assignment statements. This is because a Boolean relation containing PE variables is

automatically true in off PEs (as per 6.3.1 in the GLYPNIR Programming Manual), but is not affected by the MODE if it does not contain PE variables.

PE variables appear in Boolean expressions chiefly in relations. For Boolean assignments with PE relations,

```
BOOLEAN  B1,B2,B3;
PINT  P1,P2,P3;
B1 = P1 GEQ P2 OR B2 AND P1 LEQ P3;
B2 = B1 AND B3;
```

becomes

```
DCL(B1,B2,B3)(0:63) BIT(1);
DCL(P1,P2,P3)(0:63) FIXED BIN;
DO QQ=0 TO 63;
   IF MODE (QQ) THEN DO;
   QQRE1(QQ) = P1(QQ) >= P2(QQ);
   QQRE2(QQ) = P1(QQ) <= P3(QQ);  END;
ELSE   QQRE1(QQ),QQRE2(QQ) = TRUE;
B1(QQ) = QQRE1(QQ) | B2(QQ) & QQRE2(QQ);
B2(QQ) = B1(QQ) & B3(QQ);
```

where QQRE1 and QQRE2 are declared at the end of the generated PL/I as

```
DCL (QQRE1,QQRE2)(0:63) BIT(1);
```

BEGIN

A BEGIN becomes a DO/*BEGIN*/ unless it is followed by declarations.

Declaration Statements

PL/I attributes correspond to the different GLYPNIR types as follows:

GLYPNIR	PL/I	
REAL	FLOAT BIN(21)	single precision
REAL	FLOAT BIN(53)	double precision (see 4.2.3)
INTEGER	FIXED BIN(31)	
BOOLEAN	BIT(1)(0:63)	

In addition, for PREAL or PINT types the dimensionality attribute "(0:63)" (actually 0:#PES) is appended.

The %%C EXTERNAL option (see 4.2.2) causes the "EXT" attribute to be appended.

For subroutine statements, declarations are added for the entry point in front of the subroutine and for the formal parameters after the subroutine declaration. Thus,

```
PREAL SUBROUTINE S(PCPOINT V1, PREAL P1, CINT OUT C1);
```

becomes,

```
DCL SSS ENTRY ((*,*) FLOAT BIN, (*) FLOAT BIN,
               FIXED BIN, (0:63) FLOAT BIN):
SSS: PROCEDURE (V1, P1,C1,QQFRP);
DCL V1(*,*) FLOAT BIN, P1(*) FLOAT BIN, C1 FIXED BIN,
    (S,QQFRPR) (0:63) FLOAT BIN:
```

Note the addition of the fourth parameter for this sample function subroutine. The result is returned in QQFRPR, so that a function may be called more than once per statement. Throughout the function subroutine, references to S (the original function name) are left unchanged. Then, at the end of the function subroutine, S is assigned to QQFRPR. Non-function subroutines do not require the extra parameter, and do not have their name prefixed by SS.

IF (and ELSE), FOR, and WHILE Statements

These three statements are all characterized by a controlling Boolean expression (abbreviated cbe), and generate similar code. Their general GLYPNIR form is

keyword cbe DO statement

For IF, the 'DO' is replaced by 'THEN'. For the ELSE part of the IF, the cbe is implicitly the negation of the cbe following the IF keyword. There are two main variations, depending on the presence of PE and Boolean variables in the cbe versus a cbe with only CU variables.

Only CU Variables in the cbe. The IF-ELSE requires no translation. The WHILE becomes

```
QQW1:  IF cbe THEN DO;
          statement;
          GO TO QQW1;
          END;
```

A FOR cbe must contain non-CU variables.

PE or Boolean Variables in the cbe. The form for all is

```
QQM1 = MODE;  /* An array assignment statement to save the mode */
QQW1:  /* the label only for WHILE statements */
DO QQ = 0 TO 63;
   IF MODF(QQ) THEN
      QQIFBE1(QQ),/* for IFs*/ MODE(QQ) = cbe
      END;
IF ANY (MODE) THEN DO: /* only for WHILE statements */
statement,
GO TO QQW1: END; /* only for WHILEs */
/* the following DO and statement2 present only for ELSE
statements */
   DO QQ = TO 63;
      IF QQM1(QQ) THEN MODE(QQ) = ¬QQIFBE1(QQ);
      ELSE MODE(QQ) = FALSE;
      END;
statement2  /* statement body of ELSE */
MODE = QQM1  /* restore original mode */
```

NAMING CONVENTIONS

With one exception, all user-declared names of variables are unchanged in the PL/I. The one exception is function subroutines. These have the characters SS prefixed to their names. (This need not be done by the user for separately-compiled function subroutines, because the translator does it automatically.)

The following shows the various temporary variables and labels created by the translator, as well as a brief description of their types and uses. Small "x" represents an integer which, for each name, increases from one by one to make each instance of a name unique (where necessary). Note that all names start with "QQ". Users should thus avoid QQ except to achieve special effects by using one of the following names. '(0:63)' should be interpreted as '(0:#PES)'.

Name	Type	Use
QQ	FIXED BIN(16)	"PE" index for generated iterative DOs.
QQAPx	(0:63) FLOAT BIN(21)	Actual parameters that are complex PE expressions are evaluated outside the subroutine call and assigned to temporaries QQAPx.
QQFRBx	(0:63) BIT(1)	Boolean-type function subroutines return their multiple values to a final added argument of this form.
QQFRCIx	FIXED BIN(31)	CU-type function subroutines return their value to a final added argument of this form (mainly for consistency with multiple value PE functions). Type INTEGER.
QQFRCRx	FLOAT BIN(21)	Similar to QQFRCIx, but type REAL.
QQFRPIx	(0:63) FIXED BIN(31)	PE-type function subroutines return their multiple values to a final added argument of this form. Type INTEGER.
QQFRPRx	(0:63) FLOAT BIN(21)	Similar to QQFRPIx, but type REAL.
QQIFBEx	(0:63) BIT(1)	Preserves the controlling Boolean expression of an IF for use by a possible matching ELSE.

Name	Type	Use
QQMx	(0:63) BIT(1)	A mode "stack" for preserving the mode before an IF, FOR, or WHILE if the controlling Boolean expression contains PE or Boolean variables.
QQ#PES	FIXED BIN(16)	Number of PEs being simulated from user-parameter #PES.
QQREx	(0:63) BIT(1)	Because of the different MODE treatment necessary for Boolean assignment statements containing PE relations versus those without, these relations are evaluated separately and assigned to temporaries QQREx.

The GLYPNIR functions listed below are all supplied by the load module GLYSUBS, which must be link edited with the compiled PL/I resulting from a translation. 'x' will be 'R' for REAL or 'I' for INTEGER, depending on the argument type.

Name	Type	Use
QQEVERY	function BIT(1)	Function to evaluate Boolean quantifier EVERY.
QQSOME	function BIT(1)	Function to evaluate Boolean quantifier SOME.
QQPORx	function FLOAT BIN	'ORs' together all values of its PE argument to produce a CU variable.
QQGRABx	function	See GPM 10.10.
QQMAXx	function	See GPM 10.9.1.
QQMINx	function	See GPM 10.9.2.
QQROWSx	function	See GPM 10.11.

Appendix B

SELECTED PL/I COMPILER AND EXECUTION MESSAGES

This appendix discusses some of the common messages that may arise during compilation or execution of the generated PL/I. It also describes how to trace problems from the message back to the GLYPNIR.

PL/I COMPILER MESSAGES

All PL/I messages appear at the end of the PL/I listing and fall into one of four levels: WARNINGS, ERRORS, SEVERE ERRORS, and TERMINAL ERRORS.

In GLYPLIT applications, TERMINAL messages usually indicate a compiler error, or an implementation restriction exceeded. In the first case, the only thing to do (before IBM arrives) is to twiddle the PL/I by changing the GLYPNIR. (The CHECK condition prefix and the optimization level of the compiler are often associated with terminal errors. Removing CHECKS, and/or including 'OPT=0' in the PL1.PARM often cures them.) Implementation restrictions are discussed in Appendix J of the PL/I (F) Programmers' Guide [3]. In this case, the solution is to break up the GLYPNIR so that the restriction is not exceeded. (Exceeding the number of blocks restriction causes TERMINAL message IEM0071I and requires separating the GLYPNIR into one or more separately-compiled routines. See 2.9.3.)

SEVERE messages are usually the result of syntactically incorrect PL/I. If a GLYPLIT message was not generated during translation for the statement in question, the translator has generated bad PL/I and should be reported to the GLYPLIT distributor. Usually, however, translator messages have also been generated and once the GLYPNIR is corrected, the PL/I problems will cease.

ERROR messages may indicate syntax errors (in which case the solution is the same as for SEVERE messages) or may refer to certain implementation restrictions. In particular, ERROR message IEM2867I will be generated if any external (separately-compiled) subroutine or variable name exceeds seven characters (see the note at the end of 4.2.2). As

long as the first four characters and the last three form a unique name, this will cause no problems (although it will require a COND=(8,LT,GLYPLIT) override in the LKED step, because a return code of 8 will be returned).

In most cases, there should be no TERMINAL, SEVERE, or ERROR messages. WARNING messages, however, may be frequently generated for data and parameter conversions and other conditions. These should always be checked, especially data conversions, as they may reveal unexpected problems.

User-supplied PL/I may, of course, cause messages at all levels.

COMPILED CODE EXECUTION MESSAGES AND TRACING PROBLEMS BACK TO THE GLYPNIR

PL/I execution messages are of the form

IHEmmmI message text IN STATEMENT nnn NEAR OFFSET aaa

FROM ENTRY POINT ccc

where

mmm identifies the message;
nnn gives the PL/I statement number (if the STMT option is included as described in 5.2, card 11);
aaa is an address (usually of little interest unless it is outside of the program address space, in which case PL/I's error recovery has failed);
ccc gives the PL/I subroutine name containing the offending statement number;† and
message text will briefly describe the error (e.g., OVERFLOW, ZERODIVIDE, PROTECTION VIOLATION, etc.).

†The main program will be called MAIN if no label is prefixed to the first BEGIN (see 2.2.10). Also, since GLYPLIT prefixes all function subroutine names with 'SS' (see 2.9.1), if an error occurs inside of a function subroutine named FUNC, then ccc will be SSFUNC.

The message text and the message explanation, which may be found in Appendix K of the PL/I (F) Programmers' Guide [3], will usually define the nature of the error.

The error may be traced back to the GLYPNIR as follows:
The PL/I statement number (nnn in the message format above) identifies the PL/I statement. Then, given the PL/I listing, the sequence number in columns 77-80 of the card image contains the corresponding GLYPNIR card number shown on the GLYPNIR listing during translation. The particular GLYPNIR construct on the card may usually be identified from the offending PL/I statement text.

Two caveats: the sequence number and card number may not always agree exactly, so a little matching of the GLYPNIR and the PL/I text may be necessary; and if the entry point ccc is a GLYPLIT execution library subroutine (like QQROWSUx or QQMAX), then, since PL/I does not provide a traceback, the offending GLYPNIR must be identified by examining all of the named subroutine calls in relation to the user-generated output.

Note that, for debugging purposes, conditions like ZERODIVIDE may be disabled with condition prefixes as described in 3.4. Similarly, if PROTECTION VIOLATION or other addressing problems occur, SUBSCRIPTRANGE may be enabled to find the usual cause of these problems.

Appendix C

GLYPLIT MESSAGES

The messages listed below are produced by GLYPLIT during a translation and are written on SYSPRINT intermixed with the GLYPNIR listing. Messages fall into one of four levels as follows:

WARNING: Indicates a syntax or other error, which was probably corrected acceptably. Generates RETURN CODE = 4.

ERROR: Indicates a syntax or other error, which may (rarely) have been corrected. Will usually cause a PL/I message at the ERROR level. Generates RETURN CODE = 8.

SEVERE: Indicates a syntax error, for which no correction was attempted, or a translator internal table overflow. PL/I generated probably also syntactically incorrect at the SEVERE level. GLYPNIR input text usually skipped up to the next BEGIN, END, or SEMICOLON. Generates RETURN CODE = 12.

TERMINAL: Usually not a syntax error, but indicates a GLYPLIT internal consistency check failed. Please report to GLYPLIT distributor. Suggestions are given with each message below for rearranging code to attempt to get around problem. Generates RETURN CODE = 16.

Messages listed on SYSPRINT have the format:

```
******    level:  message text      ** msgno **
```

where "level" is one of the four levels discussed above, and "msgno" is an integer which uniquely identifies the message. If a message must be continued on more than one line, then lines after the first will have a msgno of 0.

In the list of messages below, each message is given in the format

```
msgno L  message text
              explanation and/or suggestions if necessary
```

where "L" is an abbreviation for the level of the message: W,E,S,and T, respectively. In the message text, "ccc" represents a character string filled in from the GLYPNIR, and "nnn" represents an integer.

GPM refers to the GLYPNIR Programming Manual [1].

1 S PROGRAM DID NOT START WITH <ID:|> BEGIN OR A SUBROUTINE DECLARATION.
A main program must start with a BEGIN, optionally preceded by an ID; or a subroutine declaration if it is a separately translated subroutine (see 2.9.3). "MAIN:BEGIN" was assumed.

2 S UNRECOGNIZABLE SYNTAX.
This message may be generated for rather trivial errors in statements that, at first glance, look correct. It is frequently the result of a missing semicolon, a misspelled keyword, not having a space between a keyword and a '(' or ')', etc.

3 S NEST OF LOOP, FOR, WHILE, IF, AND/OR ELSE STATEMENTS TOO DEEP LIMIT IS 10.
These statements may be nested, i.e., completely contained within each other, up to a maximum depth of 10. An 'IF' and its corresponding 'ELSE' count as 1. The solution is to break up the structure by using statement labels and GO TO statements.

4 S BOTH SIDES OF AN ASSIGNMENT STATEMENT MUST BE OF THE SAME TYPE--BOOLEAN OR ARITHMETIC.
See GPM 7.1.

5 S LOGICAL END OF PROGRAM FOUND BEFORE END OF GLYPNIR INPUT. ERROR SCAN CONTINUES.
This means that at least one more 'END' has been parsed successfully than 'BEGIN'. BEGINs may be lost due to syntax errors in a statement just before a BEGIN.

6 S ccc NOT DECLARED AS A LABEL, BUT WAS DECLARED ON CARD NUMBER nnn.
Generated either for a GO TO with an undeclared destination or for 'ccc:' if ccc is not a declared label. In the later case, may be a result of using a ':' instead of a ';'.

7 T UNKNOWN QQCTRL = nnn.
Report to GLYPLIT distributor. May sometimes be cured by removing %%C cards.

8 S THE BODY OF A FOR, LOOP, OR WHILE STATEMENT MAY NOT BE LABELED.
See GPM 8.3, 8.1, and 8.4, respectively. These statements all require setting up iterative counts or other tests and a branch directly to the body would miss the settings.

9 T FCSTI ≠ 0 AT SUBROUTINE CALL, FCSTI = nnn.
Report to GLYPLIT distributor. May sometimes be cured by removing any nested function calls or function calls as parameters.

10 E LABEL ccc MUST BE DECLARED IN THE INNERMOST BLOCK IN WHICH IT IS USED.
The PL/I code will probably execute correctly, but this is a GLYPNIR error as stated in the GLYPNIR Programming Manual.

12 S LOOP STATEMENT HEADERS MUST CONTAIN ONLY CU VARIABLES.
See GPM 8.1. The iterative FOR is the correct substitute if PE variables are required, but it is not implemented, so the user must do the incrementing and testing with assignment, IF, and GO TO statements.

13 S ONLY 'FOR ALL' STATEMENTS ARE IMPLEMENTED.
The two unimplemented kinds of FOR statements are the iterative FOR (for which the user may substitute his own IF, assignment, and GO TO statements), and the FOR ANY, which is obsolete (and for which the user may usually substitute a FOR ALL).

15 S 'IF' STATEMENTS NESTED TOO DEEPLY--LIMIT IS 10.
Replace some of the nested IF structure with statement labels and GO TO statements.

16 S THE ELSE IS NOT MATCHED BY A PRECEDING IF.
This means thst at least one more 'ELSE' has been successfully parsed than 'IF'. It may result from a syntax error in a preceding IF. In a nest of IF/ELSE statements, this message may appear at the end of the nest rather than with the actual unmatched ELSE.

17 S FORMAL PARAMETER 'ccc' DOES NOT HAVE A LEGAL TYPE. CREAL ASSUMED.

The TYPE of a formal parameter must be declared with the parameter in the subroutine declaration, e.g., TYPE PREAL for P1 in SUBROUTINE S(PREAL P1). See GPM 9.1.

18 S TYPE ccc IS NOT IMPLEMENTED. PREAL ASSUMED.

In GLYPLIT, only types CREAL, CINT, PREAL, and PINT are allowed.

19 S DIMENSION MUST BE BETWEEN 0 AND 2049. 100 ASSUMED.

Note that in GLYPNIR, subscripts always run from 0 to the dimension given.

20 S SUBROUTINE DECLARATIONS NESTED TOO DEEPLY. LIMIT IS 5.

The solution is to declare subroutines at same level instead of within each other.

21 E ALL DECLARATIONS MUST BE AT THE HEAD OF THE BLOCK.

The generated PL/I will probably be correct because this is not a PL/I restriction, but it is a GLYPNIR error.

22 S 'ccc' MUST BE A CU EXPRESSION.

23 S 'ccc' MUST BE A PE EXPRESSION.

24 S IF A FORMAL PARAMETER IS DECLARED TO BE A POINTER, THEN THE ACTUAL PARAMETER MUST BE AN UNSUBSCRIPTED VECTOR NAME.

Subroutine declarations are the only circumstance in GLYPLIT where pointers are permissible. As noted in Chapter 5 of the GLYPNIR Programming Manual, this is the only mechanism for passing unsubscripted vectors, i.e., whole arrays, to subroutines. To pass a subscripted vector parameter, i.e., a single element of a vector (e.g., a row for PE vectors of a scalar for CU vectors), the formal parameter declaration is just CREAL, PREAL, CINT, or BOOLEAN. See also 2.9.

25 S ccc HAS BEEN DECLARED AS A VECTOR AND MUST BE SUBSCRIPTED.

This occurs for an unsubscripted vector name as an actual parameter when the corresponding formal parameter is not of type POINTER. See also message 24.

26 T FUNCTION_CALL_END_S, BAD PROC-SYNTAB-PT.
Report to GLYPLIT distributor. May sometimes be cured by re-arranging function calls.

27 S ccc NOT DECLARED AS A FUNCTION.
An attempt was made to use ccc as a function subroutine, but it was not declared with a type, e.g., PREAL SUBROUTINE ccc and so it may only appear as a subroutine call, i.e., 'ccc;'.

28 E NUMBER OF PARAMETERS IN CALL TO ccc DOES NOT AGREE WITH NUMBER DECLARED.
Although GLYPNIR permits missing parameters, GLYPLIT does not.

29 W ROUTES WHERE THE <MODE PATTERN> IS NOT EMPTY MAY CAUSE PROBLEMS.
See this GLYPLIT Manual 2.10.2.

30 S IMP AND EQV ARE NOT IMPLEMENTED. PLEASE USE PL1 BOOL FUNCTION.
This refers to the Boolean operators IMP and EQV. The BOOL function may be used as follows:
B1(QQ)IMP B2(QQ) ≡ BOOL(B1(QQ),B2(QQ), '1101')
B1(QQ)EQV B2(QQ) ≡ BOOL(B1(QQ),B2(QQ), '1001')
Because B1 and B2 represent Boolean variables, their PL/I declarations will be Bx (0:63) BIT(1). In GLYPNIR, the statement might be
B3=B1 IMP B2;
But in the translated PL/I the user must supply the DO loop. Thus the complete user-supplied substitute for the above statement would be:

```
%%B  DO QQ=0 TO 63;
%%       B3(QQ)=BOOL(B1(QQ),B2(QQ),'1101'); END;
```

31 W THE FIRST CHARACTER OF A HEX CONSTANT SHOULD BE ≤ '9'.
See GPM 4.5.1.

32 S HEX CONSTANTS HAVE A MAXIMUM LENGTH OF 16 SIGNIFICANT HEXITS.
A hex constant may contain 17 characters only if the first is '0'. It may never contain more than 17 characters.

34 S ccc NOT DECLARED AS A VECTOR. MAY BE CAUSED BY USING < FOR LSS.
This occurs if an attempt is made to subscript a name that has not been declared as a vector. Note: this will occur if "<" is used for "less than", e.g., P1<P2, since "<" also delimits subscripts. Use "LSS" for "less than".

35 S SUBSCRIPTS ARE NESTED TOO DEEPLY, LIMIT IS 5.
E.G., A<B<C<D<E<F<G>>>>>> is one too many.

36 S ccc NOT DECLARED.
Will occur if the declaration of ccc contained a syntax error or if the BEGIN-END blocks structure is in error. Note that in GLYPNIR, all variables must be declared. Also, any global variable used in a subroutine declaration, i.e., used in the body of the subroutine but declared in the main program, must be declared *before* the subroutine.

37 S ccc HAS BEEN DECLARED AS A VECTOR AND MUST BE SUBSCRIPTED.
This occurs for unsubscripted vector names appearing other than as actual parameters. See also message 25.

39 E REACHED END OF GLYPNIR INPUT BEFORE LOGICAL END OF PROGRAM.
Means successfully parsed at least one more 'BEGIN' than 'END'. An END may have been lost if it was not preceded by a semicolon and/or a syntax error occurred.

40 W PROGRAM SHOULD END WITH 'END:'.
Final period is probably missing.

41 S ccc ALREADY DECLARED IN THIS BLOCK ON CARD nnn.
A variable may be redeclared in an inner block, but not twice in the same block. BEGIN-END structure probably wrong.

42 W MSG MAY NOT BE SET GREATER THAN 3.
The user PARM parameter MSG may not be set so as to ignore TERMINAL messages. 3 assumed.

43 W THE FOLLOWING PART OF THE INPUT PARAMETER WAS UNRECOGNIZABLE: 'ccc'
This refers to the PARM parameter from the JCL EXEC card. ccc shows the offending part of the parameter.

44 E MUST HAVE 1≤#PES≤999. 64 ASSUMED.

45 W IF GLY PARM IS 2 OR 3 THEN PL1 PARM MUST BE 2 OR 3.
That is, if the GLYPNIR input is to be copied to the PLIN file, then there must be a PLIN file requested. Two is added to parameter PL1.

46 W MUST HAVE THE FOLLOWING RELATIONSHIPS: LC,RC,CC<81; LC<RC; AND (CC<LC OR CC<RC).
That is, all source must be on an 80 or less column card image; the left margin must be to the left of the right margin; and any carriage control column must be outside of the source margins. Assumed LC=1, RC=72, CC=0.

47 E APPARENTLY MISSING SEMICOLON. SEMICOLON INSERTED BEFORE ccc...
This message rarely appears, because the translator seldom can parse the input if semicolons are missing.

48 E TRUNCATION HAS OCCURRED. MAYBE JUST NAME>31 CHARACTERS. CHECK PL1 GENERATED.
During translation, many pieces of the GLYPNIR must be placed into temporary locations, all of which have fixed maximum sizes. If the translator attempts to store a piece in a temporary location too small for it, then the piece will be truncated, i.e., all characters past the limiting size will be lost and this message will be issued. Names of length > 31 characters are truncated in this manner (as stated in 2.2.4), and as long as the first 31 characters are unique, no problems will arise. However, truncation of other syntactic constructs, e.g., actual and formal parameters, subscripts, etc., will usually result in erroneous PL/I. When this message appears, the user should check the PL/I to see if the translation of the offending statement is correct. Usually, a PL/I compiler error message will also result. The solution is to determine the truncated construct and break it into smaller pieces.

49 W 'ELSE' MUST BE PRECEDED BY A STATEMENT OR BLOCK AND NOT BY A SEMICOLON.
The form "IF <Boolean expression> THEN s_1; ELSE s_2" is incorrect. No semicolon should be present. s_1 and s_2 must be single statements or BEGIN-END blocks.

50 E LABELS ON ELSE STATEMENTS MUST FOLLOW THE 'ELSE', NOT PRECEDE IT.
See GPM 8.5.

51 S DECLARATIONS MAY NOT BE LABELED.

52 S FUNCTION CALLS NESTED TOO DEEPLY. LIMIT IS 5.

53 W TRIED TO TAKE ddd OF CU value 'ccc'. USED CU VALUE SHOWN DIRECTLY.
ddd is either MAX, MIN, or ROWSUM. See GPM 8.9 and 8.11.

54 W ALREADY INSIDE ccc DCL BLOCK. NEW 'START ccc' CARD IGNORED.
Refers to %%C option.

55 W NOT INSIDE ccc DCL BLOCK. 'END ccc' CARD IGNORED.
Refers to %%C option.

56 S STATEMENT TOO LONG. NUMBER OF CHARACTERS BETWEEN SEMICOLONS MUST BE LESS THAN 257 (NOT COUNTING ALL BUT ONE LEADING AND TRAILING BLANKS ON EACH CARD).
See 2.2.5 in this manual.

57 W EXTRA SEMICOLON IGNORED.
Extra semicolons are illegal in GLYPNIR.

58 S 'WHILE' STATEMENTS NESTED TOO DEEPLY. LIMIT IS 10.
Replace WHILE structure with IF and GO TO statements, but note limit on depth of a structure made of any combination of IFs, WHILEs, and FORs is also 10.

59 S A SUBROUTINE MAY NOT CONSIST OF A SINGLE IF STATEMENT. PLEASE ENCLOSE THE IF STATEMENT IN A BEGIN-END BLOCK.
This is a translator restriction.

60 S RE PARAMETER NUMBER nnn: EITHER BOTH THE FORMAL AND ACTUAL PARAMETERS MUST BE BOOLEAN OR NEITHER MAY BE. NO CONVERSION IS POSSIBLE.
If a formal parameter is of type Boolean, then the argument must be also, and vice versa.

61 W 'EXTERNAL' ATTRIBUTE NOT ALLOWED IN SUBROUTINE DECLARATION. IT HAS BEEN IGNORED.
The word EXTERNAL is used as part of the entry declaration only in the routine which *calls* the subroutine. See Sec. 4.2.4.

62 W ALREADY INSIDE BLOCK OF COMMON DECLARATION. SECOND COMMON STATEMENT WITHOUT INTERVENING ENDCOMMON STATEMENT IGNORED.

63 E NOT IN BLOCK OF COMMON DECLARATIONS. I.E., NO COMMON STATEMENT TO MATCH ENDCOMMON STATEMENT. ENDCOMMON IGNORED.

Appendix D

SUMMARY OF JCL PARM OPTIONS

The following options are input as a list, PARM='list', on the GLYPLIT EXEC card (see card 2, 5.1). They are fully described in 4.1 under the subsections shown.

Subsection	Name	Range	Default	Meaning
4.1.1	#PES	1-999	64	Number of PEs for which code is to be generated.
4.1.2	GLY	1	1	List GLYPNIR on SYSPRINT.
		2		List GLYPNIR on PLIN.
		3		List GLYPNIR on both SYSPRINT and PLIN.
4.1.3	PL1	1-3	2	Same as GLY but refers to generated PL/I.
4.1.4	PAGES	>0	50	Maximum number of SYSPRINT pages allowed.
4.1.5	MSGS	>0	50	Maximum number of messages allowed on SYSPRINT.
4.1.6	MSG	1-3	1	Minimum significant message level.
4.1.7	LC	1-80	1	Source text left margin.
	RC	1-80	72	Source text right margin.
	CC	0-80	0	Carriage control character column.
4.1.8	BCRC	1-64	1	Number of bits to repeat for Boolean constants if #PES > 64.

Appendix E

SAMPLE JCL LISTINGS

The following five annotated figures show sample JCL for:

Figure	Contents
E.1	Translation
E.2	Translation and compilation
E.3	Translation, compilation, link edit, and execution
E.4	Translation and compilation of a separate routine
E.5	Translation and compilation of a final separate routine followed by link edit and execution of several routines

```
//TRANS  JOB  PARAMETERS
//GLYPLIT EXEC PGM=GLYPGO,REGION=300K
//STEPLIB DD DSN=LGLYPLIT,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=3129)
//PLIN DD DSN=&&TEMP,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=SYSDA,
//     DISP=NEW,SPACE=(TRK,(1,1))
//GLYPNIR DD *
        .
        .
   GLYPNIR INPUT DECK
        .
        .
/*
```

Fig. E.1--Translation

```
//TRANCOMP  JOB  PARAMETERS
//GLYPLIT EXEC PGM=GLYPGO,REGION=300K
//STEPLIB DD DSN=LGLYPLIT,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=3129)
//PLIN DD DSN=&&TEMP,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=SYSDA,
//     DISP=(NEW,PASS),SPACE=(TRK,(1,1))
//GLYPNIR DD *
        .
        .
   GLYPNIR INPUT DECK
        .
        .
//PL1 EXEC PL1LFC,REGION.PL1L=250K,PARM.PL1L='SIZE=999999',
//PL1L.SYSIN DD DSN=*.GLYPLIT.PLIN,DISP=OLD
/*
```

Fig. E.2--Translation and Compilation

Note:

1. The REGION.PLIL and SIZE subparameter assume a large program (>350 cards of GLYPNIR).

```
//TRANCLG  JOB  PARAMETERS
//GLYPLIT EXEC PGM=GLYPGO,REGION=300K,PARM='#PES=4'
//STEPLIB DD DSN=LGLYPLIT,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=137,BLKSIZE=6584)
//PLIN DD DSN=&&TEMP,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),
//    DISP=(NEW,PASS),UNIT=SYSDA,SPACE=(TRK,(1,1))
//GLYPNIR DD *
% THIS PROGRAM READS IN A 4 BY 4 ARRAY (NOTE THAT THE
% NUMBER OF PES=4) AND PRINTOUTS THE ARRAY, THE MAXIMUM
% VALUE IN THE ARRAY, THE MINIMUM, AND THE AVERAGE OF ALL
% ELEMENTS IN THE ARRAY.
%
DEMO: BEGIN
PREAL VECTOR X<3>;  % ALL VECTORS HAVE A LOWER BOUND OF 0.
CREAL MAXIMUM, MINIMUM, SUM, TEMP;
CINT I;
%
%%B GET FILE(SYSIN) EDIT(X) (4 (4 F(5,0),SKIP));
%
MAXIMUM = MAX(X<0>);  MINIMUM = MIN(X<0>);  SUM = ROWSUM(X<0>);
LOOP I=1,1,3 DO
BEGIN
     TEMP = MIN(X<I>);
     IF TEMP LSS MINIMUM THEN MINIMUM = TEMP;
     TEMP = MAX(X<I>);
     IF TEMP GTR MAXIMUM THEN MAXIMUM = TEMP;
     SUM = SUM + ROWSUM(X<I>);
END;
%
%%B PUT FILE(SYSPRINT) EDIT('THE ARRAY =',X,'THE MAXIMUM =',
%%          MAXIMUM,'THE MIMIMUM =',MINIMUM,'THE AVERAGE =',
%%          SUM/16.0)
%%     (A,4 F(8,2),3 (SKIP,X(11),4 F(8,2)),
%%          3 (SKIP(2),A,F(8,2)));
END.
//PL1 EXEC PL1LFCLG,COND.PL1L=(9,LT,GLYPLIT),
//    TIME.GO=(,10),REGION.GO=100K
//PL1L.SYSIN DD DSN=*.GLYPLIT.PLIN,DISP=OLD
//LKED.SYSLIB DD DSN=LGLYPLIT,DISP=SHR
//  DD DSN=SYS1.PL1LIB,DISP=SHR
//GO.SYSIN DD *
2.   3.   4.   5.
6.   7.   1.   8.
9.   16.  10.  11.
12.  13.  14.  15.
```

Fig. E.3--Translation, Compilation, Link Edit, and Execution

Notes: 1. This is a complete sample. With only minor changes to the JCL (the JOB card and the data set named LGLYPLIT), the new user may run this as an aid to getting started.

2. The purpose of the codes are stated in the comments heading the program.

3. For those unfamiliar with PL/I input/output, the PUT EDIT statement at the end should be helpful.

```
//MAIN  JOB  PARAMETERS
//GLYPLIT EXEC PGM=GLYPGO,REGION=300K,
//     PARM='#PES=4'
//STEPLIB DD DSN=LGLYPLIT.DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=3129)
//PLIN DD DSN=&&TEMP,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=SYSDA,
//     DISP=(NEW,PASS),SPACE=(TRK,(1,1))
//GLYPNIR DD *
MAIN: BEGIN                              %  SAMPLE MAIN PROGRAM
%%C EXTERNAL
PREAL P2:
%%C SUBR
SUBROUTINE S(PREAL P1);
P2 = 7.0;
IF BOOLEAN(07FFFFFFFFFFFFFFFE(16)) THEN S(P2-5.);
%%B  PUT FILE(SYSPRINT) DATA(P2);
END.
//PL1 EXEC PL1LFC.
//PL1L.SYSLIN DD DSN=OBJLIB(MAIN),DISP=(NEW,CATLG),
//     DCB=BLKSIZE=3200,UNIT=2314,VOL=SER=SERNUM,
//     SPACE=(TRK,(10,10,20))
//PL1L.SYSIN DD DSN=*.GLYPLIT.PLIN,DISP=OLD
```

Fig. E.4--Translation and Compilation of a Separate (Main) Routine

Notes:
1. The PARM parameter on the GLYPLIT EXEC card specifies 4 PEs.
2. Note the setups (%%C EXTERNAL and %%C SUBR cards) for the separately compiled subroutine S (see Fig. E.5).
3. The PLIL.SYSLIN card saves the object code generated by the compiler on a new partitioned data set--OBJLIB--under the member name MAIN.

```
//SUBR  JOB  PARAMETERS
//GLYPLIT EXEC PGM=GLYPGO,REGION=300K,
//     PARM='#PFS=4'
//STEPLIB DD DSN=LGLYPLIT,DISP=SHR
//SYSPRINT DD SYSOUT=A,DCB=(RECFM=VBA,LRECL=125,BLKSIZE=3129)
//PLIN DD DSN=&&TEMP,DCB=(RECFM=FB,LRECL=80,BLKSIZE=800),UNIT=SYSDA,
//     DISP=(NEW,PASS),SPACE=(TRK,(1,1))
//GLYPNIR DD *
SUBROUTINE S(PREAL P1);              %  SAMPLE SUBROUTINE
BEGIN
%%C EXTERNAL
PREAL P2;
P2 = SQRT(P2+P1);
END.
//PL1 EXEC PL1LFCLG,
//     COND.PL1L=(5,LT,GLYPLIT),
//     TIME.GO=(,10),REGION.GO=100K
//PL1L.SYSLIN DD DSN=OBJLIB(S),DISP=OLD
//PL1L.SYSIN DD DSN=*.GLYPLIT.PLIN,DISP=OLD
//LKED.SYSLIB DD DSN=LGLYPLIT,DISP=SHR
//  DD DSN=SYS1.PL1LIB,DISP=SHR
//LKED.SYSLIN DD DSN=*.PL1L.SYSLIN,DISP=OLD
//  DD *
 INCLUDE OBJLIB(MAIN,S)
//LKED.OBJLIB DD DSN=OBJLIB,DISP=SHR
/*
```

Fig. E.5--The Subroutines and Execution

Note:

1. The execution of the MAIN routines and subroutines will print P2(0) = 7.0 P2(1) = 3.0 P2(2) = 3.0 P2(3) = 7.0 on SYSPRINT.

Appendix F

SUMMARY OF IMPORTANT GLYPLIT RESTRICTIONS

For a more complete discussion, see the subsection indicated in parentheses after each restriction.

1. NO CODE statements, and therefore no ILLIAC IV Assembly Language (ASK), are permitted.

2. In RTL and RTR, a non-empty <mode pattern> is *ignored*. (2.10.2)

3. Only types PREAL, CREAL, PINT, CINT, and BOOLEAN are implemented. (2.4.1)

4. Subscript delimiters may be < > as well as [ ]. (2.2.1)

5. Iterative FOR statement not implemented. (2.8.5)

6. Limit of 256 significant characters between semicolons. (2.2.5)

7. Identifiers limited to 31 characters. (2.2.4)

8. Arithmetic literals: '(16)' is the only explicit base specifier permitted. (2.4.5.1)

9. DEBUG statement not implemented. (2.8.7)

10. SHIFT and REVOLVE functions take only Boolean expressions as parameters. (2.10.1)

11. Empty or missing parameters in a subroutine call are not allowed. (2.9.1)

REFERENCES

1. Lawrie, D. H., *GLYPNIR Programming Manual*, Department of Computer Science, University of Illinois, Urbana, Illinois, August 27, 1970.

2. *PL/I (F) Language Reference Manual*, IBM Corporation, Order No. GC28-8201-3, 4th ed., June 1970.

3. *PL/I (F) Programmer's Guide*, IBM Corporation, Order No. GC28-6594-6, 7th ed., June 1970.

4. *IBM System 360, Job Control Language Reference*, IBM Corporation, Order No. GC28-6704-0, 1st ed., June 1970.

5. *IBM System 360, Job Control Language User's Guide*, IBM Corporation, Order No. GC28-6703-1, 1st ed., June 1970.

6. Baer, D. M., *Subroutines in the GLYPNIR Compiler*, ILLIAC IV Document 257, January 15, 1972.